Constructions pour la cryptographie à bas coût

Sebastien Duval

▶ **To cite this version:**

Sebastien Duval. Constructions pour la cryptographie à bas coût. Cryptographie et sécurité [cs.CR]. Sorbonne Université, 2018. Français. NNT : 2018SORUS078 . tel-01900290v2

HAL Id: tel-01900290
https://inria.hal.science/tel-01900290v2

Submitted on 24 Feb 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE DE DOCTORAT DE SORBONNE UNIVERSITÉ

Spécialité

Informatique

École doctorale Informatique, Télécommunications et Électronique (Paris)

Présentée par

Sébastien DUVAL

Pour obtenir le grade de

DOCTEUR de SORBONNE UNIVERSITÉ

Constructions pour la cryptographie à bas coût

soutenue le 3 octobre 2018

devant le jury composé de :

Mme. Anne Canteaut	Inria de Paris	Directrice
M. Gaëtan Leurent	Inria de Paris	Directeur
M. François-Xavier Standaert	Université Catholique de Louvain	Rapporteur
M. Pierre-Alain Fouque	Université Rennes 1	Examinateur
M. Gregor Leander	Ruhr-Universität Bochum	Examinateur
M. Jérémy Jean	ANSSI	Examinateur
Mme. Christina Boura	Université de Versailles	Examinatrice
M. Damien Vergnaud	Sorbonne Université	Examinateur

rapportée par

M. François-Xavier Standaert	Université Catholique de Louvain
M. Thomas Peyrin	Nanyang Technological University

Financé par
l'École Doctorale en
Informatique,
Télécommunications
et Électronique
(Paris)

Équipe-Projet
SECRET
Inria de Paris,
2 rue Simone IFF,
75 012 Paris

Remerciements

On a pour coutume de garder le meilleur pour la fin, je dérogerai ici à cette règle puisque je commence par mes remerciements.

L'expérience inoubliable de cette thèse aurait été bien morne sans l'ambiance irremplaçable de l'équipe SECRET. L'atmosphère chaleureuse et la bonne humeur ambiante sont à coup sûr un moteur de création d'idées. Mes pensées vont tout d'abord à mes directeurs de thèse, Anne Canteaut et Gaëtan Leurent. Tous deux ont su encadrer ma thèse avec enthousiasme, partageant avec moi leur expérience et leur intelligence. À leurs côtés, je me suis senti progresser dans un cadre cordial, tout en jouissant d'une grande liberté de recherche. Merci à tous les deux de m'avoir donné cette chance et de m'avoir accompagné depuis mon stage.

Je remercie aussi François-Xavier Standaert et Thomas Peyrin d'avoir endossé le rôle fastidieux de rapporteurs de cette thèse, et Christina Boura, Damien Vergnaud, François-Xavier Standaert, Gregor Leander, Pierre-Alain Fouque et Jérémy Jean d'avoir accepté de faire partie de mon jury.

La recherche ne trouve de sens que dans le partage d'idées et dans une ambiance chaleureuse. À ce titre, un immense remerciement s'impose envers tous ceux qui ont contribué à rendre ces années de thèse extraordinaires. Merci à tous les membres de l'équipe SECRET : Adrien, Anaïs, André, André, Andrea, Anirudh, Anne, Anthony, Antoine, Audrey, Aurélie, Christelle, Christina, Daniel, Ferdinand, Florian, Gaëtan, Ghazal, Grégory, Irene, Jean-Pierre, Joëlle, Julia, Kaushik, Kévin, Léo, María, Mariem, Mathilde, Matthieu, Nicky, Nicolas, Pascale, Rémi, Rodolfo, Sristy, Thomas, Tim, Valentin, Valentin, Virginie, Vivien, Xavier et Yann. J'aimerais épiloguer sur chacun de ces noms, mais il faut bien parfois que la raison l'emporte. Il me paraît néanmoins essentiel de remercier à nouveau Anne, qui n'a pas seulement assumé le rôle de directrice de thèse, mais aussi celui d'une cheffe d'équipe exceptionnelle qui permet à chacun d'exprimer son plein potentiel. Il convient évidemment de rendre ici éloge à l'encadrement administratif et au soutien moral de Christelle, sans qui l'équipe serait dans un bien triste état.

Je souhaiterais aussi remercier tous les gens avec qui j'ai eu l'occasion de travailler, que ce soit dans le cadre de séminaires, de réunions ou dans un cadre moins formel, notamment les participants des ANR BLOC et BRUTUS, et tous les jeunes cryptographes avec qui j'ai eu des échanges amicaux, je pense en particulier à Christof, Pierrick, Colin, Anthony, Zakaria, Dahmun, Vincent, Victor, Romain, Francesco et Ilaria (j'ai conscience d'en oublier bien trop au moment où j'écris ces lignes).

Il en est un dans l'équipe dont le mérite de m'avoir supporté pendant trois interminables années ne saurait être tu plus longtemps. Je pense bien évidemment à Jean-Pierre, le roi de la mitraillette bulgare, rouspéteur diplômé, mon meilleur ennemi. Tu es digne de ma reconnaissance (pour une fois). Merci pour tant de bons moments, en particulier aux mots-fléchés et au babyfoot (c'est toujours agréable d'avoir quelqu'un à qui on met des raclées et qui en redemande).

Je remercie au passage toute l'équipe pour ces moments de détente partagés aux mots-fléchés [1], au babyfoot [2] et autour d'un bon verre.

Du fond du cœur, je tiens à rendre honneur au bureau C201, mes co-bureaux sans qui rien n'aurait été pareil. Mes pensées vont en premier lieu à Virginie, qui a su prouver des qualités d'écoute, de patience et de tempérance infinies à l'égard des garnements ignares que nous étions, Yann et moi, et qui a su me guider, m'accompagner et me soutenir tout au long de cette thèse, malgré la distance. Merci pour ton amitié, merci pour ces moments partagés, merci d'avoir supporté mes instants de folie douce, tes encouragements ont compté plus que je ne saurais l'exprimer.

Merci évidemment à Yann aux côtés de qui j'ai entrepris l'aventure de la thèse et avec qui j'espère cheminer à nouveau. Ensemble, nous avons progressé en tant qu'étudiants, en tant que chercheurs et en tant que personnes, débattant toujours, parfois de tout, souvent de rien. Merci pour cette entraide de tous les jours, pour ces moments de rigolade, de travail studieux et pour ta complicité. Si tu perds le goût des bonnes choses aux Pays-Bas, il y aura toujours une bière pour toi dans mon frigo. Plus que tout autre chose, n'oublie jamais : l'important, c'est le beau jeu. Ces guignols infâmes n'ont pas la moindre chance : on a les points de style [3].

Il existe pourtant dans ce bureau un triste sire, un tortionnaire éhonté de la race féline dont il m'arrive de douter qu'il mérite un éloge. Lorsqu'on dédie son temps à mettre des chats en boîte, n'est-il pas ironiquement logique de finir mis en boîte soi-même ? Et pourtant Xavier, je n'ai pas le cœur à te taquiner cette fois, car ces remerciements, tu les mérites finalement. Sans toi, je serais demeuré dans mon ignorance crasse vis-à-vis de l'informatique quantique, j'aurais manqué d'une oreille pour entendre mes plaintes bref, je me serais bien ennuyé. Courage pour la fin de ta thèse, je ne doute pas que tout ira pour le mieux.

Bien évidemment, parmi mes co-bureaux essentiels, il me faut remercier la première, Julia, qui, bien que travaillant à des manigances farfelues [4], n'en a pas pour le moins un cœur d'or. Tu m'as accueilli et tu as su créer une atmosphère dans laquelle je me suis tout de suite épanoui, et sans aucun doute tu as contribué à me confirmer mon envie de faire de la recherche.

Je remercie mes camarades de cryptographie symétrique [5], je remercie les colons du bureau Tapdance et les autochtones de la Terra Incognita, à l'autre bout du corridor, qui ont développé un langage étrange à base de signes ésotériques, et dont les peintures rupestres dépeignant des codes aux mesures inconsidérées n'ont rien à envier à mes Papillons (hormis les couleurs chatoyantes).

J'aimerais remercier pêle-mêle Grégory, Valentin, Audrey, Joëlle, Julia et Virginie de leur accueil lorsque je suis arrivé tout timide dans l'équipe, mon ami Nicky pour des discussions éclairantes, Léo dont j'admire le travail, Thomas pour ses cours de jardinage endiablés, les séminaires des universités de Versailles, Limoges et Rennes et le séminaire de Télécom ParisTech pour leur invitation, les équipes de l'Université Catholique de Louvain et de la Nanyang Technological University pour leur accueil et, de manière plus générale, j'aimerais saluer la convivialité ambiante dans la communauté.

Chacun est essentiel, chacun a son immense mérite [6], et chacun a contribué à joncher ces années de moments de convivialité sans prix.

1. Faites-moi plaisir, inventez les mots justes à ma place.
2. Un sport fait pour me plaire : autant d'énergie dans les mots que dans le jeu.
3. Tout est dit.
4. Je n'oublie pas la preuve de NP-complétude de prouver la NP-complétude de l'approximation d'un problème que tu m'avais faite lire.
5. Même ceux qui font de la cryptographie [post-]quantique.
6. Même si pour l'un ou l'autre ce mérite est discutable (j'me comprends, j'me comprends).

Je remercie Hugues Randriambololona et Hoeteck Wee d'avoir accepté de participer à mon comité de suivi.

Je tiens aussi à remercier certains de mes enseignants de mathématiques qui m'ont donné le goût de cet art, je pense en particulier à Jean-Luc Leone, Jean Pian, Nicolas Pouyanne et Brigitte Chauvin.

Je remercie François-Xavier Standaert et toute l'équipe de cryptographie de l'Université Catholique de Louvain qui m'accueilleront bientôt en post-doctorat.

Finalement, je tiens à remercier mes proches. Tout d'abord mes amis qui ont su m'accompagner et me soutenir durant les années passées, je pense bien sûr à François et Seijilo, à Gabriel, Romain, Oriane, Julien, Adrien, Matthieu et Yohann.

Et par-dessus tous, je tiens à remercier ma famille. Mes parents, Françoise et Dominique, qui ont su me soutenir malgré les doutes et les difficultés de la vie, mes frères et sœur, Timothée, Kévin, Alexandre et Laureline, sans qui la vie serait bien terne, et tous les autres membres de ma famille qui m'ont soutenu et encouragé, Denise, Michel, Marie-Thérèse, Jean, Laurent, Élisabeth, Dominique, Paulette, David, Edlira, Jean-Philippe, Nathalie, Jean-Louis, Sylviane, Thomas, Héloïse, Baptiste, Léa, Lucas, Quentin, Candice, Anthony, Angelina, et bien sûr Marc qui restera pour moi un exemple. Je ne pense pas avoir besoin de le dire, ce qui ne m'en empêchera pas : je vous aime tous du fond du cœur.

Table des matières

Introduction

La cryptographie vise à permettre la transmission d'information en garantissant la confidentialité du message, son authenticité et l'identité de son expéditeur. À ce titre, elle est présente dans les cartes à puce ou dans la plupart des échanges via Internet ; c'est la cryptographie qui permet de sécuriser les achats en ligne, les cartes de crédit, les envois d'e-mails...

Aujourd'hui, la cryptographie fait face à un nouveau défi : les objets connectés. Difficiles à sécuriser car leurs ressources sont limitées, ces objets se multiplient, offrant à d'éventuels attaquants une marge de manœuvre grandissante. Récemment, des problèmes sont survenus dans des objets connectés à cause d'un manque de sécurité. Par exemple, les clefs de voiture de l'entreprise Volkswagen avaient un signal d'ouverture et de fermeture des voitures trop peu sécurisé, et de nombreuses voitures ont ainsi été pillées de leur contenu. Il devient donc nécessaire de développer des solutions cryptographiques pour ces environnements aux moyens limités, et ce fut le sujet de ma thèse : réduire les coûts de la cryptographie.

Durant cette thèse, j'ai eu deux objectifs principaux. Le premier fut de développer une compréhension d'ensemble de la cryptographie symétrique. Dans ce sens, j'ai exploré plusieurs pistes : les chiffrements par blocs et leurs composantes, les chiffrements à flot et les MAC, tant du point de vue d'un concepteur que de celui d'un attaquant, sans hésiter à travailler à haut niveau sur les preuves de modes de MAC et à bas niveau sur les circuits binaires.

Ceci m'amène à mon second objectif : apporter mes propres contributions au domaine en me concentrant sur la problématique du bas coût. Bien comprendre les différents aspects de la cryptographie symétrique m'a permis d'identifier les choix qui ont été effectués dans les décennies précédentes, et de les revoir avec en tête un but différent : réduire les coûts sans baisser la sécurité.

Dans ce rapport, je commence par introduire la cryptographie symétrique au chapitre 1, avec une focalisation sur les chiffrements par blocs. Au chapitre 2, je m'intéresse à la construction de boîtes-S structurées, par des réseaux de Feistel et de type MISTY [CDL16] avec une optique pratique en premier lieu, puis par des Papillons avec un but plus théorique [CDP17]. Le chapitre 3 est une étude de l'autre composante majeure des chiffrements par blocs : la matrice de diffusion. Dans ce travail, nous développons un algorithme de recherche de matrices MDS à bas coût [DL18b], dont les résultats sont à ce jour les matrices les plus efficaces de la littérature. En adoptant le point de vue d'un attaquant, le chapitre 4 identifie un point faible dans une construction de chiffrement à flot, FLIP, imaginé pour des contraintes de bas coût dans une application de chiffrement complètement homomorphe hybride [DLR16]. Le chapitre 5 a pour but d'étudier les MACs, qui servent à l'authentification des messages. Dans ces travaux [DL18a], nous améliorons les preuves de sécurité des constructions de fonctions de hachage universelles lorsque des composantes sont bijectives, et nous développons un nouveau MAC, XPMAC, particulièrement efficace sur des micro-contrôleurs 32 bits. Enfin, le chapitre 6 introduit une discussion en prenant du recul sur ces études, et donne quelques exemples de perspectives.

Mes publications

[CDL16] Anne Canteaut, Sébastien Duval et Gaëtan Leurent. “Construction of Lightweight S-Boxes Using Feistel and MISTY Structures”. In : *SAC 2015*. Sous la dir. d’Orr Dunkelman et Liam Keliher. T. 9566. LNCS. Springer, Heidelberg, août 2016, p. 373–393.

[CDP17] Anne Canteaut, Sébastien Duval et Léo Perrin. “A Generalisation of Dillon’s APN Permutation With the Best Known Differential and Nonlinear Properties for All Fields of Size 2^{4k+2}”. In : *IEEE Trans. Information Theory* 63.11 (2017), p. 7575–7591.

[DL18a] Sébastien Duval et Gaëtan Leurent. “Lightweight MACs from Universal Hash Functions”. In : *En soumission* (2018).

[DL18b] Sébastien Duval et Gaëtan Leurent. “MDS Matrices with Lightweight Circuits”. In : *IACR Transactions on Symmetric Cryptology* 2018.2 (2018), p. 48–78. issn : 2519-173X.

[DLR16] Sébastien Duval, Virginie Lallemand et Yann Rotella. “Cryptanalysis of the FLIP Family of Stream Ciphers”. In : *CRYPTO 2016, Part I*. Sous la dir. de Matthew Robshaw et Jonathan Katz. T. 9814. LNCS. Springer, Heidelberg, août 2016, p. 457–475.

Chapitre 1

Panorama de la cryptographie symétrique

La Faune et la Flore

Ce chapitre a pour but d'introduire les différents outils cryptographiques utilisés actuellement pour le chiffrement et la signature de messages en exhibant une cohérence et une compréhension de l'ensemble de ce domaine. Une compréhension des objectifs, des hypothèses et des choix qui ont motivé la conception de ces primitives permettront par la suite d'en déterminer les limitations et d'identifier des pistes pour les surmonter.

1.1 Introduction à la cryptographie symétrique

Cette section a pour but d'introduire le concept de cryptographie symétrique (par opposition à la cryptographie asymétrique), d'en définir les différents acteurs et les familles de solutions apportées pour permettre des échanges sécurisés.

1.1.1 Brève généalogie de la cryptographie

Avant toute chose, un rapide historique s'impose.

1.1.1.1 Cryptographie antique

La plus ancienne trace de chiffrement connue date du XVI$^{\text{ème}}$ siècle avant J.-C. [Kah96]. Il est toutefois probable que la cryptographie soit aussi ancienne que le langage lui-même, puisque la cryptographie est la science du secret, et qu'il y a toujours eu des secrets.

1.1.1.2 Chiffrements par substitution

Chiffrement. Jusqu'au XX$^{\text{ème}}$ siècle, la majeure partie de cryptographie utilisée est le chiffrement par substitution. Il s'agit de remplacer chaque lettre ou groupe de lettres par d'autres.

Deux exemples parmi les plus connus sont le chiffrement de César, utilisé par les armées romaines, et le chiffrement de Vigenère, datant de 1586.

Le chiffrement de César (chiffrement monoalphabétique) consiste à choisir un secret, la *clef* de chiffrement K, qui est un nombre entre 0 et 25. Cette clef correspond à un décalage de l'alphabet. Ainsi, toute lettre de l'alphabet sera décalée de K positions, modulo

26 [1]. Pour déchiffrer, il suffit de connaître la clef de déchiffrement (ici égale à la clef de chiffrement), et à faire le décalage dans le sens inverse.

Le chiffrement de Vigenère est un chiffrement polyalphabétique bien plus robuste, dont la clef est un mot de l'alphabet. Il s'agit de décaler les lettres du message clair mais, suivant la position de la lettre dans le message, le décalage sera différent (ce qui revient à avoir plusieurs alphabets décalés et à choisir l'alphabet suivant la position de la lettre dans le message).

Cryptanalyse. La cryptanalyse est l'analyse théorique d'un chiffrement dans le but de déchiffrer le texte sans connaître la clef de déchiffrement. Le premier exemple de cryptanalyse remonte à Al Kindi au IX$^{\text{ème}}$ siècle [AK92].

Il s'agit d'une analyse de fréquence, un type d'attaque générique qui fonctionne particulièrement bien sur tous les chiffrements monoalphabétiques. L'idée est de remarquer que dans toute langue, certaines lettres ou certains groupes de lettres apparaissent plus fréquemment que les autres. En français par exemple, le 'e' est la lettre la plus fréquente. En faisant une analyse statistique de la fréquence d'apparition des lettres dans un message chiffré, on est capable de faire des hypothèses qui peuvent permettre de retrouver des informations sur le texte clair. Par exemple, on peut associer à la lettre la plus fréquente du chiffré la lettre la plus fréquente du langage, à la deuxième lettre la plus fréquente dans le chiffré la deuxième lettre la plus fréquente du langage, et ainsi de suite. Même si l'hypothèse ne fonctionne généralement pas du premier coup, il suffit généralement de peu d'adaptation pour être capable de déchiffrer entièrement un message chiffré, ce qu'on appelle « casser » le chiffrement, par une analyse de fréquence (du moins pour les chiffrements monoalphabétiques).

1.1.2 Kerckhoffs, Turing, Shannon et la cryptographie moderne

Jusqu'aux années 1970, la cryptographie était utilisée majoritairement par des militaires comme outil stratégique pour communiquer malgré les interceptions ennemies. Un principe majeur avait déjà été formalisé dans ce contexte par Auguste Kerckhoffs [Ker83] : tout le secret d'un algorithme de chiffrement doit reposer dans la clef de déchiffrement, car on ne peut pas garantir que l'algorithme, *i.e.* la manière de chiffrer avec une clef variable, ne soit jamais découverte. Ce principe reste aujourd'hui central en cryptographie.

Lors de la seconde guerre mondiale, les forces alliées et les forces de l'Axe utilisaient des télécommunications et étaient bien conscientes de l'importance de garder secrètes les manœuvres militaires. Ainsi, chaque armée inventa son propre système de chiffrement. L'un des chiffrements principaux de l'Axe, baptisé Enigma, fut d'abord cassé par les Polonais, puis fut mis-à-jour et, malgré tous les efforts des Alliés, les moyens à disposition ne permettaient pas de casser ce chiffrement.

1.1.2.1 Un nouvel outil : l'ordinateur

Ceci fournit l'argument qu'il manquait pour financer la fabrication du premier ordinateur, Colossus, capable d'une puissance de calcul digne de ce nom en suivant les idées d'Alan Turing. Grâce à ce nouvel outil, les Alliés purent casser Enigma et accéder aux communications de leurs ennemis.

Plus important pour la cryptographie moderne, ce nouvel outil qu'était l'ordinateur montrait les limites des chiffrements polyalphabétiques et démontrait la nécessité d'une approche théorique quant au chiffrement. Les chiffrements devenaient en effet trop complexes à comprendre sans fondements théoriques poussés, et il devint donc nécessaire de baser la cryptographie sur des mathématiques, avec pour espoir de pouvoir prouver qu'un

1. Rappelons que l'opération « x modulo n » consiste à prendre le reste de la division de x par n, ou de manière équivalente, à poser $n = 0$.

chiffrement ne pourrait pas être cassé. Par ailleurs, l'ordinateur est aussi un outil qui permet d'implémenter des chiffrements plus complexes.

1.1.2.2 Shannon : théorie de l'information et naissance de la cryptographie moderne

Dans le même temps, Claude Shannon développait sa théorie de l'Information, qui allait devenir la théorie fondamentale utilisée dans les ordinateurs. Mais mis à part les ordinateurs, Shannon s'intéressait à la transmission de l'information, et c'est tout naturellement qu'il s'intéressa à la transmission de secrets. En 1949, il publia un article considéré comme la pierre angulaire de la cryptographie moderne, dans lequel il définit une notion de sécurité inconditionnelle et démontra qu'il était possible d'atteindre un tel niveau de sécurité – à un coût non-négligeable. Il établit aussi des critères nécessaires pour qu'un chiffrement soit sécurisé.

1.1.3 Les années 1970 : cryptographie civile et nouveaux horizons

La décennie 1970 marqua un tournant majeur pour la cryptographie.

1.1.3.1 Cryptographie civile

À force de lutte politique et juridique, il fut accepté que la recherche en cryptographie puisse être rendue publique. Ceci permit à la cryptographie d'être utilisée par les civils, tels que les banques et les entreprises désireuses de préserver leurs secrets industriels. Dans le même temps démarrait ce qu'on a baptisé parfois Ère de l'information ou Ère de la communication, dans laquelle une grande quantité d'information circule par des canaux matériels difficiles à sécuriser. Ceci pose aujourd'hui d'importants problèmes de vie privée, pour laquelle la cryptographie est un outil adapté pour garantir des échanges sécurisés entre particuliers.

D'autre part, à travers la recherche publique, il devint possible d'entreprendre des collaborations entre les chercheurs à un niveau international, ce qui mena en particulier à l'élaboration de la plusieurs normes cryptographiques, bien que la première norme de chiffrement, DES, ait été imaginée par IBM.

1.1.3.2 Le premier standard de chiffrement : DES

Le premier standard de chiffrement, baptisé Data Encryption Standard, ou DES, fut établit en 1975 et possédait déjà toutes les caractéristiques et la structure des chiffrements actuels. S'il est aujourd'hui caduc, c'est que les moyens des attaquants se sont nettement améliorés, en particulier en termes de puissance de calcul, et que de nouvelles manières de casser des chiffrements ont été imaginées pour attaquer le DES.

Avoir une norme est essentiel : cela permet d'avoir une solution accessible lorsque l'on n'est pas un expert du domaine, et cela permet aux experts d'étudier en détails ce standard pour en identifier les faiblesses, car dans la cryptographie comme ailleurs, avoir des avis extérieurs avisés est essentiel pour détecter les faiblesses.

1.1.3.3 Cryptographie asymétrique

Dans le même temps, une toute nouvelle perspective fut envisagée. Depuis des millénaires, il avait semblé impossible d'échanger des informations secrètement sans avoir convenu d'un protocole et d'un secret, la clef, au préalable. Or, en 1976, Diffie et Hellman donnèrent leurs noms au premier protocole dit *asymétrique* [DH76].

Jusqu'alors, il était entendu que la clef de chiffrement et la clef de déchiffrement étaient identiques, d'où le nom de protocole *symétrique*. Mais ceci n'est pas un prérequis : pour envoyer un message chiffré, Alice n'a besoin que de la clef de chiffrement, et pas de la

clef de déchiffrement. Bob, lui n'a besoin que de la clef de déchiffrement. Or, la seule information qui ait besoin de rester secrète est la clef de déchiffrement : qu'importe si quelqu'un connaît la clef de chiffrement, cella-là ne sert qu'à chiffrer des messages. La seule clef à protéger est la clef de déchiffrement.

Ainsi, Alice n'a pas besoin de connaître une information secrète : elle n'a besoin que de la clef de chiffrement. Bien évidemment, il faut néanmoins que la clef de chiffrement dépende de la clef de déchiffrement. Il faut donc que Bob génère la clef de chiffrement à partir de sa clef de chiffrement, de telle sorte que personne ne pourra faire l'opération inverse (retrouver la clef de déchiffrement, secrète, à partir de la clef de chiffrement, publique). Il faut pour cela que faire l'opération inverse soit extrêmement coûteux (on parle de *fonction à sens unique*, c'est-à-dire une fonction qui est simple à calculer, mais dont l'inverse est complexe à calculer).

Or, ceci est possible, et Diffie et Hellman ont proposé un tel protocole asymétrique.

L'idée du protocole d'échange de Diffie-Hellman est le suivant. On ignorera ici le besoin d'authentifier Alice et Bob. Supposons qu'Alice et Bob veuillent faire en sorte d'obtenir une information secrète en commun (une clef pour des chiffrements ultérieurs par exemple), en faisant en sorte qu'une potentielle attaquante, Ève, ne puisse pas lire leur communication. Le protocole est le suivant :

1. Alice choisit un nombre premier p et un générateur du corps $\mathbb{Z}/p\mathbb{Z}$ g.
2. Alice choisit un nombre a et calcule $A = g^a \bmod p$.
3. Alice envoie à Bob p, g et A (sur un canal public).
4. Bob choisit un nombre b et calcule $B = g^b \bmod p$.
5. Bob envoie B à Alice.
6. Alice calcule $B^a = g^{ab} \bmod p$, Bob calcule $A^b = g^{ab} \bmod p$.
7. Alice et Bob possèdent alors un secret en commun : $g^{ab} \bmod p$.

En revanche, le chiffrement asymétrique est plus contraignant que le chiffrement symétrique, et de ce fait, on sait faire du chiffrement symétrique à un coût moindre que le chiffrement asymétrique. On préfère donc utiliser le chiffrement symétrique dans la mesure du possible. Ce qui est fait généralement, c'est qu'Alice et Bob font un échange sécurisé asymétrique comme Diffie-Hellman dans un premier temps, dans lequel ils se partagent une information secrète K ($K = g^{ab} \bmod p$ dans le cas d'un échange Diffie-Hellman). Cette information secrète, une fois partagée, leur permet de communiquer en utilisant du chiffrement symétrique.

Le protocole de générique pour entamer un échange de messages est donc le suivant : si Alice veut envoyer un message à Bob,

1. Alice génère une clef secrète K_S qui servira à faire du chiffrement symétrique.
2. Bob génère un couple de clefs secrète et publique S et P qui serviront à faire du chiffrement asymétrique. Bob publie la clef publique asymétrique P : elle est disponible pour tout le monde.
3. Alice chiffre sa clef symétrique K_S en utilisant un chiffrement asymétrique paramétré par la clef P.
4. Bob récupère le chiffré de K_S et le déchiffre grâce à sa clef de déchiffrement S.

Ainsi, Alice a envoyé sa clef de chiffrement symétrique K_S à Bob de façon sécurisée grâce au chiffrement asymétrique. Désormais, Alice et Bob partagent une clef secrète (K_S) et peuvent donc échanger des messages de façon sécurisée en utilisant du chiffrement symétrique.

Pour schématiser, le chiffrement correspond à un cadenas virtuel. Pour envoyer un message de façon sécurisée, on met ce message dans une boîte (qu'on supposera incassable [2]),

2. Rendre cette boîte solide est le rôle du domaine intitulé « sécurité informatique ».

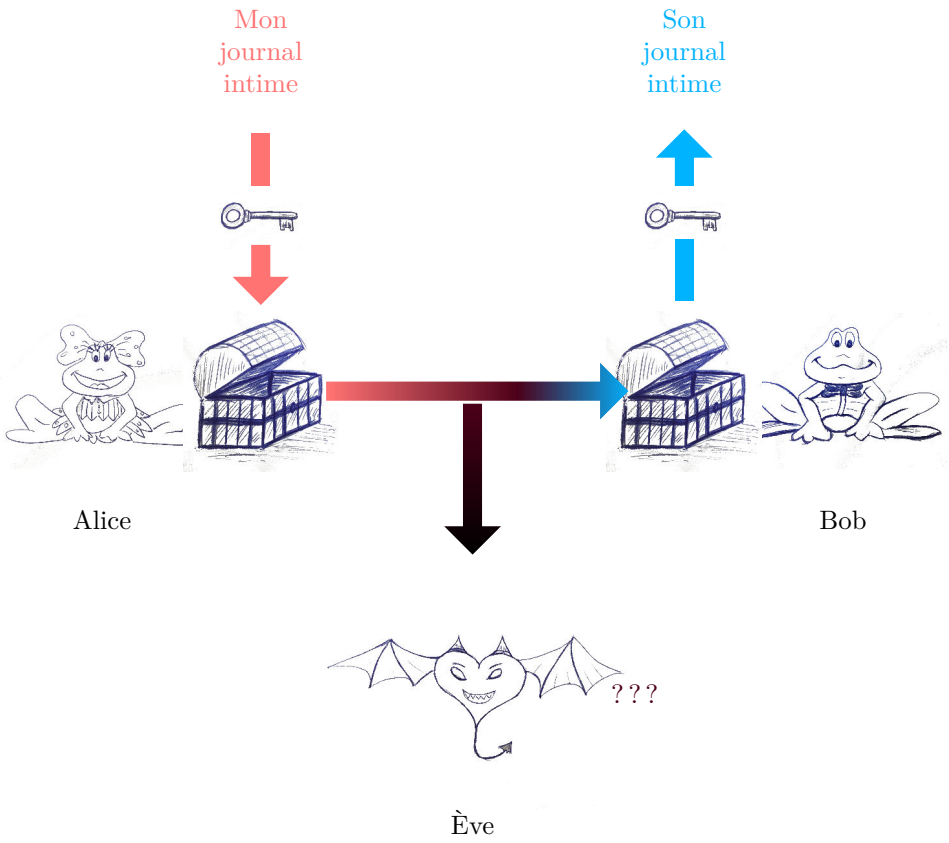


Figure 1.1 – Schéma d'un chiffrement symétrique.

et on ferme cette boîte avec un cadenas à l'aide d'une clef de chiffrement. On envoie la boîte par un canal de communication standard (par exemple la poste dans dans un univers physique, ou Internet dans un univers virtuel) à son destinataire et, puisque la boîte est fermée à clef et qu'aucun tiers ne possède la clef pour déverrouiller le cadenas (clef de déchiffrement), aucun tiers ne peut lire le message. Le destinataire, en revanche, possède la clef de déchiffrement et peut donc déverrouiller le cadenas pour retrouver le message.

Le cas du chiffrement symétrique correspond au fait qu'on verrouille et déverrouille le cadenas avec une clef identique, le cas du chiffrement asymétrique correspond au fait de verrouiller et déverrouiller avec des clefs différentes. Par exemple un cadenas qui se verrouille automatiquement lorsqu'on le ferme, mais qui nécessite une clef pour le déverrouiller.

1.1.4 Cryptographie pour les usages contemporains

Le chiffrement est une problématique ancienne de la cryptographie qui possède aujourd'hui des solutions viables.

D'autres applications de la cryptographie se sont développées depuis quelques décennies, comme par exemple l'authentification, et des problèmes pratiques se posent suivant les cas d'application qui nécessitent d'adapter la cryptographie à son cas d'usage. Je cite ici quelques cas d'usages typiques parmi beaucoup.

1.1.4.1 L'authentification par les MACs

Les MACs (Message Authentication Code) sont une autre primitive essentielle de la cryptographie symétrique. Un MAC est une famille de fonctions (à clef secrète) $A =$

$(A_k)_{k\in\mathbb{F}_2^K}$ qui associe à un message de taille quelconque une étiquette de taille fixée. L'objectif pour un MAC est qu'une attaquante ne puisse pas « forger » une étiquette, c'est-à-dire, pour une fonction aléatoire de la famille A_k, qu'elle soit incapable de trouver un couple (m, t) tel que $A_k(m) = t$ (sans avoir fait une requête à l'oracle de A pour le message m). Plus exactement, elle en est capable, mais avec une probabilité très faible de l'ordre de $\frac{1}{t}$, où t est la taille du MAC.

Sous cette condition, un MAC permet d'authentifier le contenu d'un message.

Par ailleurs, un MAC permet aussi de garantir l'intégrité d'un message, c'est à dire qu'il n'a pas été modifié durant sa transmission. En effet, si le message avait été modifié, le MAC ne correspondrait plus au message, et il serait difficile de modifier aussi le MAC pour qu'il corresponde au nouveau message sans connaître la clef. Néanmoins, le MAC possède des propriétés plus fortes que ce que requiert l'intégrité, et il est possible de garantir l'intégrité avec des fonctions plus simples (des algorithmes de hachage).

1.1.4.2 Cryptographie pour Internet

Dans les protocoles de communication par Internet, il est évident que l'échange de données peut requérir de la confidentialité (chiffrement), de l'authenticité (MAC) et de l'intégrité (hachage). Mais la situation d'Internet est très particulière, puisque ce réseau traite des quantités gargantuesques de données à chaque instant, avec nombre d'applications variées, telles que l'e-banking, les achats en ligne, la communication avec des objets connectés, les partages de contenus collaboratifs tels que Wikipédia et bien d'autres encore. Chacune de ces applications a des besoins spécifiques : les objets connectés utilisent généralement des microprocesseurs embarqués disposant de peu de ressources mais traitent peu de données, alors qu'à l'opposé, les serveurs d'Internet doivent traiter énormément de données mais disposent de gros moyens. Il est donc nécessaire d'adapter les outils cryptographiques aux besoins du contexte. Par exemple pour les serveurs, on favorisera la rapidité des chiffrements, même si c'est au détriment du coût en mémoire ou en énergie.

1.1.4.3 Cryptographie pour les microprocesseurs

À l'instar des serveurs, les microprocesseurs intégrés dans des objets aussi divers que des badges d'accès, des pacemakers ou des objets connectés nécessitent aussi de la cryptographie. Or, dans ces cas-là, la rapidité n'est généralement pas un facteur très limitant, mais la consommation d'énergie ou la taille du circuit intégré opérant la fonction sont essentiels à prendre en compte : on n'ose imaginer un chirurgien rappelant un patient au bloc pour une opération à cœur ouvert pour changer les piles du pacemaker toutes les semaines parce que les primitives cryptographiques consomment trop d'énergie.

Il est donc nécessaire de développer des solutions adaptées à chacun des usages de la vie contemporaine, ce qui implique en particulier qu'un unique standard générique optimal dans toutes les situations n'est a priori pas envisageable.

1.1.4.4 Pérennité et mises à jour

Un facteur à garder en mémoire lorsqu'il s'agit de cryptographie, c'est qu'il est nécessaire de toujours garder non pas une, mais plusieurs longueurs d'avance sur une potentielle attaquante, ce qui bien évidemment n'est pas simple. En effet, si une organisation gouvernementale stockait actuellement tout le trafic d'Internet, chiffré ou non, même sans avoir les moyens de les déchiffrer aujourd'hui, puis, dans 20 ans, avec des outils plus avancés, était en capacité de déchiffrer toutes ces informations, nul doute que certaines informations sensibles seraient ainsi compromises. Il est aussi essentiel que les primitives

cryptographiques soient mises à jour dès que possible, et pas uniquement lorsqu'elles deviennent obsolètes, puisqu'il est nécessaire de prévoir des données sécurisées non pas par rapport aux moyens actuels, mais par rapport aux moyens qui seront disponibles dans un futur proche.

Il est donc nécessaire de penser les solutions cryptographiques avec plusieurs décennies d'avance (ce qui, bien entendu, n'est pas aisé puisque l'on ne connaît pas les outils qui seront utilisés dans 20 ans).

Il est vrai pourtant que certains standards cryptographiques actuels semblent pouvoir résister indéfiniment. Quoi qu'il en soit, il n'existe aucune garantie qui affirme que les standards actuels ne seront jamais obsolètes. Il est donc nécessaire, lorsqu'on intègre de la cryptographie dans un système, de faire en sorte que ces primitives cryptographiques puissent être mises à jour (simplement si possible).

1.2 Cryptographie symétrique

Rentrons désormais dans plus de détails par rapport aux définitions et aux solutions actuelles proposées par la cryptographie symétrique.

1.2.1 Définir la sécurité : définitions informelles

En cryptographie, le premier problème est de définir la notion de sécurité, pour un chiffrement ou pour un MAC.

La définition de la sécurité des chiffrements date de Shannon en 1949, avec tout d'abord une notion très rigide de sécurité inconditionnelle, puis une notion plus réaliste de sécurité calculatoire.

La sécurité sur les MACs est définie similairement à haut niveau.

1.2.1.1 Première définition trop exigeante

Le travail de définition de la sécurité a été amorcé par Claude Shannon dans les années 1940 [Sha49], avec une première définition excessivement rigide : est considéré parfaitement sûr un chiffrement qui garantit qu'aucune personne ne possédant pas la clef secrète n'est en mesure de retrouver le message clair à partir du message chiffré, ni d'obtenir quelqu'information que ce soit sur la clef secrète ou le message. Cette définition permet à l'attaquante d'avoir à sa disposition des ressources infinies.

Shannon a prouvé que, pour satisfaire cette définition de sécurité, un chiffrement doit nécessairement utiliser une clef secrète au moins aussi grande que le message à chiffrer, et doit changer de clef à chaque envoi de message.

Mieux, Shannon exhibe un chiffrement sûr dans ce modèle très rigide : le chiffrement de Vernam, aussi appelé masque jetable, qui consiste en un XOR bit à bit du message et de la clef. Ce chiffrement est on ne peut plus simple et on ne peut moins coûteux, mais l'inconvénient majeur est que, pour envoyer secrètement une information de taille T, les deux interlocuteurs doivent déjà connaître (et consommer) une information de la même taille T. Ce n'est pas très efficace, mais ce protocole peut néanmoins être utilisé en pratique, par exemple en stockant sur un disque dur plusieurs GB de clef, à utiliser par la suite pour du chiffrement.

Ainsi, pour cette définition, un chiffrement sûr nécessite d'utiliser des quantités immenses de clefs. Tout chiffrement moins coûteux en clefs n'est pas sûr dans ce modèle dit de *sécurité inconditionnelle*.

1.2.1.2 Deuxième définition réaliste

Le premier modèle est irréaliste car il accorde à l'attaquante des ressources infinies. Plutôt que d'empêcher un attaquant de pouvoir retrouver la clef ou le message clair avec n'importe quelles ressources, la notion de sécurité plus réaliste, dite de *sécurité calculatoire*, consiste à offrir à l'attaquant des moyens considérés comme raisonnables – si l'on peut dire qu'un temps de calcul plus long que la durée de vie de l'univers est raisonnable – que ce soit en termes de temps de calcul, de mémoire, d'énergie ou autres, en partant du principe que l'attaquant dispose des meilleures ressources actuelles ou qui seront disponibles dans un futur proche.

Ainsi, on estime actuellement que faire 2^{80} opérations élémentaires [3] est à la portée de quelques acteurs majeurs comme Google ou la NSA – mais il faut vraiment qu'ils soient prêts à investir beaucoup de moyens – et est impraticable pour tout autre attaquant potentiel. Par exemple, pour exister, bitcoin nécessite actuellement de l'ordre de 2^{80} opérations élémentaires toutes les 10 minutes, infrastructure dédiée et centrales d'énergie à l'appui [4]. Pour cette raison, une sécurité calculatoire de 2^{80} est considérée à l'heure actuelle comme raisonnable, mais pas excellente.

Généralement, pour des chiffrements solides, on estime que le temps de calcul nécessaire à une attaque doit être d'au moins 2^{128} opérations élémentaires [5]. Dans le jargon, on parle parfois de « 128 bits de sécurité ».

Dès qu'on fixe une limite aux ressources de l'adversaire, il n'y a plus de preuve qu'il est impossible de chiffrer un grand nombre de bits à partir d'une clef petite.

1.2.1.3 Conditions nécessaires à la sécurité

Shannon va plus loin dans son analyse du chiffrement [Sha49]. Il établit deux critères essentiels pour qu'un chiffrement soit sûr : la diffusion et la confusion.

La diffusion est une notion relativement simple, qui dit que tout bit du message chiffré doit dépendre de tous les bits du message clair. Cette notion de « dépendance » signifie simplement que, si un bit change en entrée, le bit de sortie peut être impacté. Une telle dépendance peut être assurée par des fonctions très simples, et même aussi simples que des fonctions linéaires.

La confusion est une notion plus floue. Il s'agit d'énoncer que le lien entre le message clair et le message chiffré doit être « compliqué », pour une certaine notion de complexité mal définie. En tous les cas, une chose est sûre, c'est que le lien ne peut pas être linéaire. Il est donc nécessaire qu'une partie du chiffrement apporte de la non-linéarité, et plus généralement qu'aucune structure évidente n'apparaisse dans le chiffrement.
L'inconvénient d'une fonction non-structurée est double : une telle fonction est complexe à analyser et coûteuse à implémenter en pratique, d'autant plus si c'est une fonction sur un grand nombre de bits.

Cette étude préliminaire des contraintes apporte déjà une bonne idée de ce à quoi doit ressembler un chiffrement sûr, ce qui permet aux cryptographes d'évaluer la sécurité de leurs chiffrements et par la suite de normaliser des chiffrements considérés sûrs.

En particulier, on se rend compte qu'on aura besoin de différencier les fonctions linéaires et les fonctions non-linéaires. Pour cette raison, j'introduis ici un code couleur qui sera

3. XOR par exemple.

4. D'après l'agence Bloomberg, maintenir la sécurité de bitcoin demande entre 8 et 38 TWh par an (suivant les sources), soit entre la consommation de l'Estonie et celle du Pérou et de l'ordre de la consommation de 3,4 millions de foyers américains. D'après les estimations, il s'agirait aujourd'hui de 0.1% de la consommation d'énergie mondiale.https://www.bloomberg.com/gadfly/articles/2018-01-04/bitcoin-s-cheap-energy-feast-is-ending

5. En considérant un processeur usuel à 2^{30} opérations par seconde, il faut 10^{22} années, soit mille milliards de fois l'âge de l'univers, pour faire 2^{128} opérations.

respecté dans toutes les figures de ce document : les opérations linéaires seront en jaune, les opérations non-linéaires en bleu.

Fonction linéaire. Rappelons la définition d'une fonction linéaire : soient un corps $(K, +, \cdot)$ et U et V deux espaces vectoriels sur $(K, +, \cdot)$. une fonction $f : U \to V$ est dite *linéaire* si pour tous $x, y \in U$ et pour tout $c \in K$, $f(x + y) = f(x) + f(y)$ et $f(c \cdot x) = c \cdot f(x)$.

En particulier, sur un corps binaire $\mathbb{F}_2^n$, on utilise généralement la loi additive $\oplus$, c'est-à-dire le XOR (ou exclusif bit à bit). Donc une fonction linéaire $f : \mathbb{F}_2^n \to \mathbb{F}_2^n$ sera telle que pour tous $x, y \in \mathbb{F}_2^n$, $f(x \oplus y) = f(x) \oplus f(y)$. Trivialement, l'opération XOR elle-même est linéaire. D'autres opérations linéaires notables sont les permutations de bits (et donc en particulier les rotations de tous les bits par un décalage fixe) et les multiplications par un scalaire (*i.e.* par une matrice binaire). Pour citer quelques opérations non-linéaires usuelles, les opérations logiques binaires and et or sont non-linéaires, tout comme les additions modulaires [6] et les fonctions puissances (pour des puissances non-triviales).

1.2.2 Définir la sécurité : définitions formelles

La confusion de Shannon nécessite une notion de *complexité*.

Cette complexité est généralement définie comme de l'aléatoire.

Afin de définir formellement le chiffrement et sa sécurité, nous aurons besoin d'introduire un certain nombre d'objets : les familles de permutations, les distingueurs et les oracles.

1.2.2.1 Chiffrement

Famille de fonctions/permutations. Pour définir un chiffrement, il est nécessaire tout d'abord de définir les *familles de fonctions*. Une famille de fonctions est une fonction à deux entrées $F : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^m$, où les deux entrées sont une clef $K \in \{0,1\}^k$ et un message $x \in \{0,1\}^n$, et la sortie est un chiffré $F(K, x) \in \{0,1\}^m$. Pour tout clef K, on définit une fonction à clef F_K comme $F_K(x) = F(K, x)$ pour tout $x \in \{0,1\}^n$.

On appelle permutation une fonction bijective sur n bits, i.e. une fonction $f : \{0,1\}^n \to \{0,1\}^n$ inversible.

De même, on définit les familles de permutations : une famille de permutations est une famille de fonctions F telle que pour toute clef K, F_K est une permutation.

On appellera généralement n la taille du clair, m la taille du chiffré et k la taille de clef.

Remarquons donc que, lorsqu'on parlera de *famille*, on sous-entendra l'existence d'une clef (généralement secrète).

Définition d'un chiffrement. On appelle chiffrement une famille de permutations sur n bits, où n est la taille du clair [7].

1.2.2.2 Sécurité des chiffrements

Oracles. On appelle *oracle* pour F une primitive abstraite avec accès en « boîte-noire » à F, c'est-à-dire une primitive qui, étant donné une entrée x pour F, renvoie $F(x)$ mais sans donner de détails sur la fonction F (*i.e.* sur la façon dont $F(x)$ a été calculé).

6. On peut aussi définir $\mathbb{F}_2^n$ avec pour loi additive l'addition modulaire, qui est alors trivialement une opération linéaire.

7. En particulier, n est une variable dépendante du message. On verra plus loin les « chiffrements par blocs », dont la taille de clair est fixe.

Distingueur. La notion de sécurité des chiffrements repose sur le concept de distingueur.

Soient F et G deux ensembles de fonctions à valeurs dans $\{0,1\}^m$. L'objectif est de distinguer F de G.

On définit la fonction $\tilde{F}$ à valeurs dans $\{0,1\}^m$ par :

$$\tilde{F}(x) = f(x),\ f \xleftarrow{\$} F$$

où la notation $f \xleftarrow{\$} F$ signifie que f est tirée au hasard dans F suivant une distribution uniforme.

On appelle *distingueur* ou *adversaire* entre F et G un algorithme qui a accès à un oracle pour $\tilde{F}$ et à un oracle pour $\tilde{G}$, qui fait un appel à l'un des deux oracles choisi au hasard, et qui renvoie 1 s'il pense qu'il s'agit de l'oracle pour $\tilde{F}$ et 0 s'il pense qu'il s'agit de l'oracle pour $\tilde{G}$.

En particulier, une famille de fonctions ou une famille de permutations est un ensemble de fonctions, donc on peut définir un distingueur entre la famille de fonctions F et un ensemble de fonctions G.

De même, l'ensemble des fonctions (respectivement des permutations) à valeurs dans $\{0,1\}^m$ est un ensemble de fonctions (notons-le $\mathcal{F}$), donc on peut définir un distingueur entre $\mathcal{F}$ et un ensemble de fonctions G. On parlera souvent d'un distingueur entre G et une fonction aléatoire (reps. une permutation aléatoire), voire si le contexte est assez clair d'un distingueur de G.

Avantage. Nous sommes désormais capables de définir l'*avantage* d'un adversaire (ou d'un distingueur).

Soit un adversaire $\mathcal{A}$. On note $\mathbb{P}(\mathcal{A}^F \to 1)$ la probabilité que $\mathcal{A}$ renvoie 1 s'il accède à l'oracle pour $\tilde{F}$.

L'avantage de l'adversaire $\mathcal{A}$ pour distinguer F de G est alors :

$$\mathbf{Adv}_F^G(\mathcal{A}) = \left|\mathbb{P}(\mathcal{A}^F \to 1) - \mathbb{P}(\mathcal{A}^G \to 1)\right|,$$

Alors que F est *inconditionnellement indistingable* de G si pour tout adversaire $\mathcal{A}$,

$$\mathbf{Adv}_F^G(\mathcal{A}) = 0\,.$$

De même, F est *calculatoirement indistingable* de G si pour tout adversaire « pratique » $\mathcal{A}$ (adversaire qui a accès à une quantité « pratique » de ressources [8]),

$$\mathbf{Adv}_F^G(\mathcal{A}) \sim 0\,,$$

plus précisément, si

$$\mathbf{Adv}_F^G(\mathcal{A}) = \varepsilon\,,$$

alors $\mathcal{A}$ a un avantage ε pour distinguer F de G.

Comme dans le cas des définitions de Shannon, la définition inconditionnelle est peu réaliste, et nous considérerons généralement la définition calculatoire. On dira donc que F est *indistingable* de G, en omettant le terme « calculatoirement ».

PRF et PRP. On note PRF une famille de fonctions pseudo-aléatoires et PRP une famille de fonctions pseudo-aléatoires, c'est-à-dire qu'une PRF à valeurs dans $\{0,1\}^m$ est indistingable d'une fonction aléatoire (tirée uniformément au hasard parmi les fonctions à valeurs dans $\{0,1\}^m$), et similairement pour les PRP.

8. Qui peut faire moins de 2^{80} opérations élémentaires par exemple.

Sécurité d'un chiffrement. Formellement, considérons un chiffrement E de taille de clair n et de taille de clef k, c'est-à-dire une famille de permutations $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ avec E_K bijectives pour tout $K \in \{0,1\}^k$.

E est sûr en tant que PRF si E est indistingable d'une fonction aléatoire pour tout adversaire pratique $\mathcal{A}$, *i.e.*

$$\mathbf{Adv}_E^{PRF}(\mathcal{A}) \sim 0\,,$$

pour tout $\mathcal{A}$ pratique. En d'autres termes, un adversaire pratique aura un avantage négligeable pour distinguer E d'une fonction aléatoire.

On parlera généralement de *bits de sécurité* : E est sûr en tant que PRF jusqu'à b bits de sécurité si pour tout $i \in \{1, \ldots, b\}$, tout adversaire $\mathcal{A}$ effectuant moins de 2^{b-i} requêtes et moins de 2^{b-i} opérations élémentaires (XOR bit à bit par exemple),

$$\mathbf{Adv}_E^{PRF}(\mathcal{A}) \leqslant 2^{-i}\,.$$

Une borne usuelle en cryptographie est de dire que 128 bits de sécurité donnent une bonne sécurité.

De même, E est sûr en tant que PRP si E est indistingable d'une permutation aléatoire pour tout adversaire pratique $\mathcal{A}$.

1.2.2.3 Cryptanalyse

De l'importance de la cryptanalyse. Cela signifie qu'un chiffrement E est considéré comme sûr s'il n'existe pas de distingueur de E avec des ressources raisonnables. Or il existe une infinité de distingueurs possibles : un distingueur consiste simplement à observer une propriété de E et il existe une infinité de propriétés étudiables.

Par définition, il est donc impossible de prouver qu'un chiffrement est sûr. En revanche, il est possible de prouver qu'un chiffrement n'est pas sûr : il faut pour cela exhiber un distingueur de E en un temps raisonnable.

C'est le rôle de la cryptanalyse, qui consiste à faire une analyse théorique des primitives cryptographiques pour estimer leur résistance. Il apparaît donc que la cryptanalyse est un axe crucial, puisqu'elle c'est elle seule qui permet d'estimer la sécurité d'une primitive cryptographique, que ce soit en se plaçant du côté d'un attaquant pour tenter une attaque ou en se plaçant du côté du créateur en donnant des preuves ou des arguments de sécurité.

Ainsi, l'axe principal de la cryptographie moderne est un compromis sécurité/coût [9]. L'objectif majeur est alors de parvenir à construire des primitives avec de bons arguments de sécurité (idéalement des preuves d'intistingabilité) et un coût d'implémentation raisonnable (en particulier, une taille de clef petite par rapport à la taille de message). La cryptologie moderne développe une gamme de primitives positionnées différemment par rapport à ce compromis et spécialisées pour leurs cas d'usage.

Les modèles d'attaques sur les chiffrements. On considère usuellement plusieurs types d'attaques sur les chiffrements suivant les moyens donnés à l'attaquante. Citons les modèles les plus courants.

Dans toute ces attaques, l'objectif d'Ève est de retrouver des informations sur la clef ou sur des correspondances clair-chiffré qu'elle n'a pas testées. On estimera son avantage en considérant tous les adversaires avec des ressources raisonnables qui respectent le modèle.

On compare généralement une attaque à la recherche exhaustive de la clef : si l'attaquante teste toutes les clefs, elle peut sans peine déterminer la valeur de la clef utilisée dans le chiffrement. Ceci demande de l'ordre de 2^k chiffrements, où k est la taille de la

9. Ou de manière équivalente, risque/coût.

clef. Pour cette raison, on choisit k assez grand pour que 2^k chiffrements ne soit pas une quantité de calculs raisonnable ($k = 128$ par exemple). L'objectif de l'attaquante sera donc de trouver une attaque plus efficace que la recherche exhaustive.

Le modèle le plus contraignant pour l'attaquante est le modèle d'attaque *à chiffré connu*. Dans ce modèle, on considère que l'attaquante n'a accès qu'à des chiffrés qu'elle ne contrôle pas et doit tenter de distinguer le chiffrement d'une permutation aléatoire.

Un modèle laissant plus de liberté est le modèle d'attaque *à clair connu*, dans lequel l'attaquante a accès à des couples (clair, chiffré).

Ces deux modèles sont passifs car l'attaquante ne fait qu'observer les communications et n'interfère pas. Les modèles suivants sont des modèles actifs.

On appelle modèle d'attaque *à clair choisi* le modèle dans lequel l'attaquante a un accès en boîte-noire au chiffrement et peut donner en entrée n'importe quel message clair et observer le chiffré correspondant (et ce autant de fois qu'elle le désire).

On appelle modèle d'attaque *à chiffré choisi* le modèle dans lequel l'attaquante a un accès en boîte-noire au déchiffrement et peut donner en entrée du déchiffrement n'importe quel message chiffré et observer le clair correspondant (et ce autant de fois qu'elle le désire).

Un modèle plus théorique est celui dans lequel on accorde à l'attaquante un pouvoir sur la clef : on appelle modèle d'attaque *à clefs liées* un modèle dans lequel l'attaquante est capable d'accéder au chiffrement en boîte-noire pour plusieurs clefs en connaissant une relation entre ces clefs. En allant encore plus loin, on peut même considérer un modèle d'attaque *à clefs choisies*, peu réaliste en général, mais qui peut malgré tout apporter des analyses intéressantes.

1.2.3 Ce qui est difficile : concilier sécurité et coût d'implémentation

1.2.3.1 Que l'aléatoire s'implémente mal

L'inconvénient majeur de l'aléatoire, c'est que, par définition, il n'a pas de structure, et que par conséquent il est coûteux à implémenter. La raison est que trouver une implémentation peu coûteuse d'une fonction donnée sans structure est un problème difficile, au sens NP-difficile (comme montré par exemple pour les fonctions linéaires dans [BMP13], alors même que les fonctions linéaires sont structurées).

Ainsi, si l'on ne fait pas l'effort de générer une implémentation peu coûteuse par construction, on est ramené à implémenter une fonction par sa table des valeurs, *i.e.* stocker pour chacune des entrées la valeur de son image par la fonction. Par exemple, la table suivante est une représentation de la fonction qui envoie 0 sur 0, 1 sur 2, 2 sur 4, etc :

$$[0, 2, 4, 6, 1, 3, 5, 7]$$

Pour implémenter une telle table, il faut stocker en mémoire toutes les images. Pour une fonction de n bits, il y aura donc 2^n images à stocker, chacune représentée sur n bits, ce qui nécessite $n2^n$ bits de mémoire. Or, pour $n = 128$, on atteint 2^{135} bits de mémoire, soit bien plus que toute la mémoire informatique présente sur Terre. Il est envisageable de stocker une fonction de 8 bits, ce qui prend 2^{11} bits de mémoire, soit 256KB, mais dans certains cas d'application, on ne s'autorise à implémenter de cette manière que des fonctions de 4 bits, dont la table requiert $2^6 = 64$ bits de mémoire seulement.

1.2.3.2 La structure comme compromis sécurité-implémentation

Il est donc nécessaire de trouver un compromis entre la sécurité du chiffrement et son coût d'implémentation pour arriver à des implémentations pratiques. La solution naturelle est d'introduire de la structure dans le chiffrement pour en réduire son coût. Par exemple, l'œil avisé aura noté que la table du paragraphe précédent correspond à la fonction $x \mapsto 2x \bmod [8 = 1]$. Une manière alternative d'implémenter cette fonction est donc d'appliquer cette formule mathématique pour trouver l'image d'une entrée x. On peut aussi imaginer construire des fonctions opérant sur un grand nombre de bits (par exemple 128) en les construisant à partir de blocs plus petits (sur 4 ou 8 bits par exemple) ou de fonctions simples (linéaires par exemple). C'est le choix fait usuellement dans les chiffrements par blocs (cf. section 1.2.7).

Bien évidemment, introduire de la structure est loin d'être une solution parfaite. En effet, la structure est précisément l'opposé de l'aléatoire. Ainsi, un chiffrement très structuré sera très simple à distinguer d'une permutation aléatoire et ne sera donc pas sûr. Toute la subtilité de la cryptographie consiste alors à identifier des structures simples à implémenter, mais difficiles à distinguer de l'aléatoire.

Remarquons en passant que ce compromis entre la sécurité et le coût d'implémentation sous-entend qu'il n'y a pas de définition évidente d'un « chiffrement optimal ». Il faudra donc adapter les choix de chiffrements aux cas d'usage : si l'on peut se permettre un coût d'implémentation élevé, on pourra utiliser des chiffrements peu structurés, donc en principe plus sûrs, mais si l'environnement sur lequel on veut implémenter le chiffrement est trop restreint (une simple puce RFID par exemple, ou un micro-contrôleur comme on en trouve dans les objets connectés), il sera nécessaire d'introduire plus de structure, ce qui affaiblira potentiellement la sécurité [10].

1.2.3.3 En bref

Pour chiffrer m bits sans donner d'information ni sur le clair, ni sur la clef, une solution existe : c'est le chiffrement de Vernam, qui permet de générer n bits de texte chiffré sûr, et qui coûte de générer et de stocker n bits aléatoires.

Heureusement, il n'est pas nécessaire d'utiliser un chiffrement qu'on peut prouver sûr inconditionnellement : un tel chiffrement garantit que l'attaquante ne pourrait obtenir *aucune information* sur le clair ou la clef. Or, nous pouvons nous contenter de bien moins : il suffit que l'attaquante ne puisse pas récupérer *assez d'information* pour retrouver des informations sur le clair ou la clef, et ce avec des moyens limités.

Dès lors, il est possible d'envisager des chiffrements moins coûteux que des permutations aléatoires, en particulier, il devient envisageable de chiffrer m bits de message avec une clef de n bits, $n < m$ en utilisant un chiffrement pseudo-aléatoire. L'objectif sera alors de prouver que l'attaquante n'a pas les moyens de distinguer le chiffrement d'une PRP ou d'une PRF, et d'en déduire des bits de clef ou de clair. C'est là le rôle de la cryptographie.

S'il n'est pas possible par définition de prouver qu'un chiffrement est sûr, il est tout à fait envisageable de prouver qu'un chiffrement est distingable d'une permutation aléatoire : il suffit d'exhiber une propriété distingante. C'est là le rôle de la cryptanalyse. Faute de preuve, on considérera qu'un chiffrement est fiable si l'on ne connaît aucune attaque ni aucune propriété distingante exploitable après un effort conséquent de cryptanalyse [11].

10. On peut aussi formuler cela comme : pour un niveau de sécurité N, il sera plus dur d'atteindre ce niveau de sécurité sur un environnement restreint que sur un environnement très permissif.

11. Notons tout de même qu'il n'est pas impossible qu'on découvre un jour une faille dans un chiffrement considéré comme fiable. Ceci implique que lorsqu'on implémente un chiffrement, il doit être simple de le mettre à jour.

1.2.4 De l'importance de la cryptanalyse

Un effort conséquent est donc accordé à la cryptanalyse.

On distinguera deux types d'attaques : les attaques classiques (ou génériques) et les attaques spécifiques.

Au fil des années, il est apparu que certains types de distingueurs sont particulièrement efficaces et fonctionnent de façon générique, sur tout chiffrement, et mènent généralement à des attaques efficaces. On citera en particulier les attaques algébriques, qui exploitent un degré trop faible du chiffrement vu comme un polynôme en les bits de la clef, les attaques différentielles, qui reposent sur l'étude des propriétés de la dérivée du chiffrement et les attaques linéaires, qui tentent d'approximer le chiffrement par une fonction linéaire. Ces attaques sont puissantes et génériques, et il est donc nécessaire de vérifier que toute nouvelle proposition de chiffrement est robuste face à ces attaques classiques.

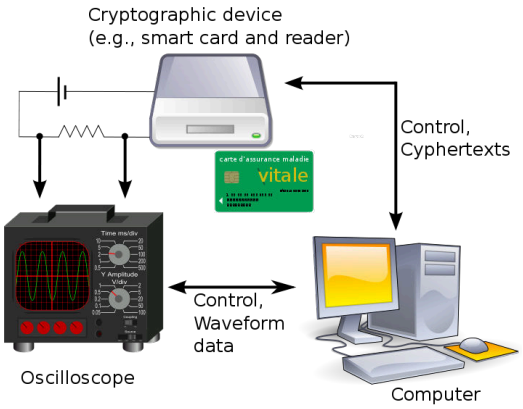
Il est aussi nécessaire de s'intéresser à chaque chiffrement individuellement, et l'on se rend compte que parfois, on est capable d'attaquer un chiffrement par une attaque spécifique. La plupart du temps, il s'agit néanmoins de raffinements des attaques classiques.

En plus de cela, de nouvelles idées de cryptanalyse émergent. En 2011 par exemple, Leander *et al.* [Lea+11] ont imaginé un nouveau type d'attaque lors de leur étude du chiffrement PRINTcipher [Knu+10]. Il s'agit d'attaques dites par sous-espace invariant. L'idée est qu'un sous-espace des clairs a pour image le même sous-espace sur les chiffrés. Ceci n'est évidemment pas attendu d'une permutation aléatoire et permet donc de distinguer le chiffrement d'une permutation aléatoire.

Ces travaux d'analyse des primitives cryptographiques sont essentiels : ils permettent de se mettre à la place d'un attaquant et de prévoir les faiblesses potentielles que pourrait trouver l'attaquant, afin de pallier ces problèmes au préalable.

Poussés par la technologie et les nouvelles idées, la cryptographie doit aussi faire face à deux nouveaux adversaires : les attaques par canaux auxiliaires et les ordinateurs quantiques.

1.2.4.1 Attaques par canaux auxiliaires : où la physique blesse



Attaques par canaux auxiliaires. Les attaques par canaux auxiliaires consistent à ne pas considérer une fonction mathématique comme un objet parfait, auquel on donne une entrée et qui donne une sortie, mais à remarquer qu'en pratique, cette fonction est implémentée, et qu'on peut observer l'implémentation pour déduire de l'information. En effet, le délai pour calculer la fonction peut révéler de l'information sur la valeur calculée [Koc96], tout comme l'énergie consommée [KJJ99], ou même les sons produits par le circuit mécanique.

De la même manière, en se branchant sur le circuit qui exécute la fonction cryptographique, on peut observer des valeurs en cours de calcul, voire même observer les messages.

L'attaquante peut même être active, par exemple elle peut enlever ou remplacer des morceaux de circuit ou provoquer des erreurs de calcul pour déduire des informations en observant la sortie. Toutes ces attaques, bien différentes de celles envisagées jusqu'alors, posent de nouveaux problèmes. Les résoudre est l'un des objectifs majeurs en cryptographie.

Un exemple de contre-mesure : le masquage. L'une des contre-mesure possibles aux attaques par canaux auxiliaires est le masquage [Cha+99 ; ISW03]. L'idée est que toute valeur intermédiaire v du calcul du chiffrement qui contient une information sensible sera divisée en plusieurs parts v_i telles que $v = \sum_i v_i$ (où la somme est une somme de XORs). Ainsi, pour retrouver l'information v, une attaquante serait obligée de retrouver tous les v_i, ce qui augmente considérablement le coût de l'attaque si l'on divise v en beaucoup de parts.

Bien évidemment, une telle contre-mesure est coûteuse. Autant il n'est pas difficile d'adapter une opération linéaire L à ce cas de partage, puisqu'il suffit d'appliquer l'opération linéaire L séparément sur les parts et que, par linéarité, $\sum_i L(v_i) = L(\sum_i v_i) = L(v)$, autant les opérations linéaires sont beaucoup moins simples à gérer.

Sans rentrer dans les détails, il faut considérer que masquer une opération non-linéaire est nettement plus coûteux que masquer une opération linéaire. On s'attend a priori à un coût quadratique en la taille de la fonction non-linéaire à masquer, et on espère (sous hypothèse de bruit et d'indépendance) une augmentation exponentielle de la sécurité. Ainsi, en vue de masquer une fonction, il est important de garder à l'esprit qu'il faut minimiser le nombre d'opérations non-linéaires.

1.2.4.2 Ordinateur quantique : mieux vaut prévenir que guérir

D'autre part, la perspective d'un éventuel ordinateur quantique dans un futur proche, quelles que soient les capacités réelles dudit ordinateur, demande que l'on soit prêt à toutes les éventualités. En extrapolant les propriétés attendues d'un ordinateur quantique, il semble aujourd'hui clair que presque l'intégralité des solutions de cryptographie asymétrique ne résisteraient pas à un ordinateur quantique. Il faut donc, en cryptographie asymétrique, imaginer de nouvelles solutions résistantes face à ce nouvel adversaire. Il s'agit de l'un des sujets de recherche principaux en cryptographie asymétrique actuellement, et c'est aussi l'objectif d'un concours actuel organisé par le NIST (National Institute of Standards and Technology), qui cherche à normaliser des primitives résistantes à un ordinateur quantique [12]. Du côté de la cryptographie symétrique, bien que les primitives utilisées n'aient pas été imaginées pour résister à un ordinateur quantique, il semble que des modifications simples – bien qu'entraînant un surcoût – permettent de se mettre à niveau dans la majorité des situations. Afin de vérifier qu'en effet, les primitives symétriques ne cèdent pas face à un ordinateur quantique, une partie des recherches actuelles est dédiée à la cryptanalyse dans un modèle où l'attaquant a accès à un ordinateur quantique.

1.2.5 La situation : les normes de chiffrement

La recherche publique et la cryptographie civile ont permis de mettre en place des normes pour le chiffrement. En cryptographie symétrique, la première norme historique est le DES (Data Encryption Standard), publié en 1977 à partir d'une conception de Horst Feistel ([Des]), aujourd'hui obsolète.

Les normes plus récentes sont issues de concours internationaux : AES pour le concours de chiffrement par blocs, eSTREAM pour le concours de chiffrement à flot, plus récemment

12. https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization

CAESAR pour le concours de chiffrement authentifié (qui tente d'apporter à la fois confidentialité et authenticité) et bientôt un concours du NIST (National Institute of Standards and Technology) pour la cryptographie à bas coût.

1.2.6 Chiffrements à flot

En informatique, les messages sont représentés comme une chaîne de bits (**bi**nary digi**t**s, ce qui signifie chiffres binaires), c'est-à-dire de chiffres dans l'ensemble $\{0, 1\}$.

1.2.6.1 Principe du chiffrement à flot

Le chiffrement à flot consiste à chiffrer le message bit par bit, au fur et à mesure que les bits sont lus. Habituellement, l'approche choisie est d'implémenter un chiffrement par masque jetable (cf. section 1.2.1.1), mais avec une suite pseudo-aléatoire générée à partir d'une clef *bien plus petite* que le message. Ainsi, on génère cette suite pseudo-aléatoire (appelée *suite chiffrante*), puis bit par bit, on applique une simple opération XOR (ou exclusif) entre le bit de suite chiffrante et le bit de message pour obtenir un bit de chiffré.

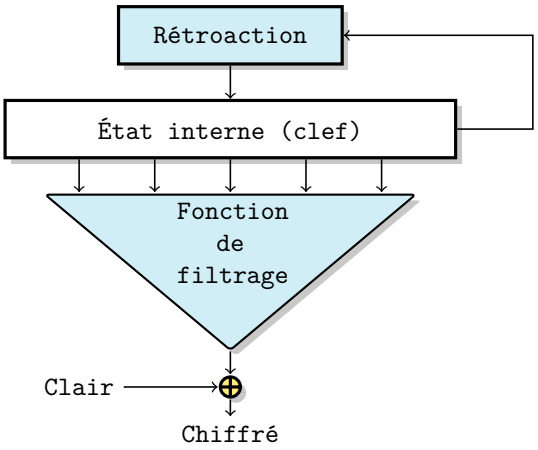


Figure 1.2 – Schéma usuel d'un chiffrement à flot.

Cette solution a plusieurs avantages : on peut imaginer des chiffrements très rapides et très peu coûteux. En particulier, on peut chiffrer un message à la volée, à flot, bit par bit, sans avoir besoin de stocker tout le message dans de la mémoire tampon.

Par construction, la sécurité des chiffrements à flot dépendra directement des propriétés de la fonction générant la suite chiffrante (pseudo-aléatoire). Formellement, il faudra que la famille de fonctions générant la suite chiffrante soit indistingable d'une PRF.

1.2.6.2 Les LFSR

L'essentiel d'un chiffrement à flot est donc un générateur pseudo-aléatoire déterministe. Une solution peu coûteuse, efficace et relativement simple à étudier est le LFSR (Linear Feedback Shift Register), registre à décalage à rétroaction linéaire, dont la sortie est une suite régie par une relation de récurrence linéaire. Il est nécessaire de choisir un LFSR approprié, mais le modèle mathématique utilisé permet de rendre ce choix simple. Le LFSR seul ne suffit pas : il n'apporte pas de confusion[13]. Il existe un grand nombre d'approches pour utiliser un ou des LFSR en rajoutant des couches de confusion supplémentaires.

13. Comme son nom l'indique, il est linéaire.

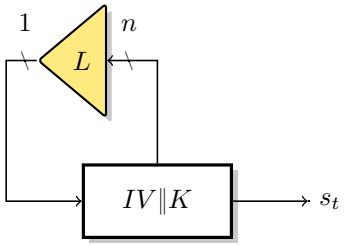


Figure 1.3 – Schéma d'un LFSR. IV est une valeur initiale (publique), K est la clef secrète. À chaque itération t, le LFSR sort un bit de suite chiffrante s_t, le registre est décalé d'un bit vers la droite et le bit de gauche du registre est mis-à-jour avec le bit de sortie de la fonction de rétroaction L.

Le concours eSTREAM [RB08] a vu candidater de nombreux chiffrements à flot, certains construits de la sorte, d'autres non, et le résultat est un portfolio de chiffrements à flot à utiliser : HC-128 [Wu08], Rabbit [BVZ08], Salsa20/12 [Ber08] et Sosemanuk [Ber+08] optimisés pour les applications logicielles, et Grain v1 [Hel+08], MICKEY 2.0 [BD08] et Trivium [CP08] optimisés pour le matériel.

1.2.7 Chiffrements par blocs

Là où le chiffrement à flot traite un message bit par bit, le chiffrement par blocs, comme son nom l'indique, les traite bloc de bits par bloc de bits. La taille de bloc est fixée à l'avance (les tailles usuelles sont de l'ordre de 64/128/256 bits).

Formellement, un *chiffrement par blocs* est donc une famille de permutations dont la taille d'entrée est fixée (contrairement à un chiffrement dont l'entrée est de taille variable).

L'avantage majeur de cette approche est qu'on peut faire dépendre le chiffré d'un bit de tous les bits du message d'entrée (ou au moins d'autant de bits qu'il y en a dans un bloc). En revanche, elle est un peu plus coûteuse et un peu moins rapide que le chiffrement à flot [14] (bien que toujours très raisonnable), mais surtout, un chiffrement par blocs ne peut pas être directement appliqué sur chaque bloc de message indépendamment (mode ECB), puisqu'on tombe alors dans le même travers, à savoir qu'il n'y a pas de diffusion entre les blocs, et en particulier, les chiffrés de deux blocs identiques seraient identiques (l'expression consacrée étant (« On voit le pingouin. ») sera ici remplacée par : « On voit le papillon. », par égard pour la thématique de cette thèse, cf. figure 1.4). Le chiffrement par blocs nécessite donc l'utilisation d'un *mode opératoire de chiffrement*, qui définit la manière de chaîner les différents appels au chiffrement par blocs (cf. section 1.2.7.1).

1.2.7.1 D'une taille fixe à une taille quelconque : les modes opératoires

En pratique, les messages ne sont pas de taille fixe, et sont généralement plus grands que 128 bits. Ainsi, il faut une façon intelligente d'utiliser un chiffrement par blocs pour chiffrer des messages de taille quelconque (supérieure à 128 bits).

La manière directe qui consiste à chiffrer le message bloc par bloc est à l'évidence catastrophique : les blocs sont chiffrés indépendamment, et donc deux blocs de message clair égaux auront des chiffrés égaux (par exemple en figure 1.4).

Il faut donc chiffrer chaque bloc différemment.

Un grand nombre de modes opératoires viables ont été développés, listons-en quelques-uns.

14. Sauf parfois pour chiffrer des messages courts.

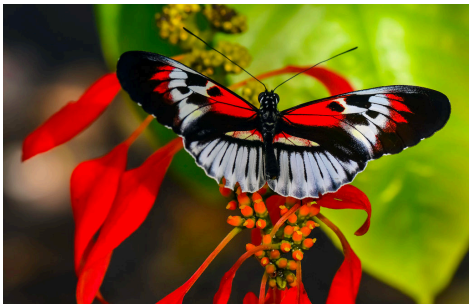


Figure 1.4(a): Une image anodine d'un papillon.

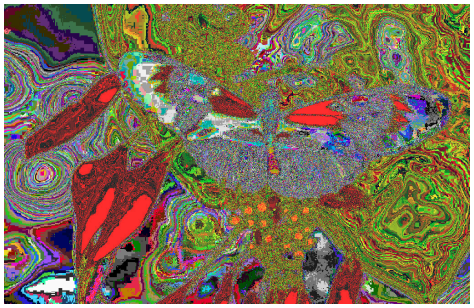
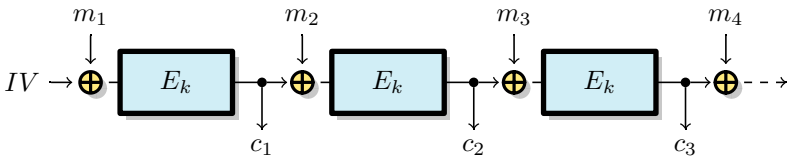


Figure 1.4(b): La même image après des modifications identiques sur chaque pixel, ce qui permet de vérifier l'adage : on voit le papillon.

Figure 1.4 – Exemple d'un mauvais mode opératoire (ECB).

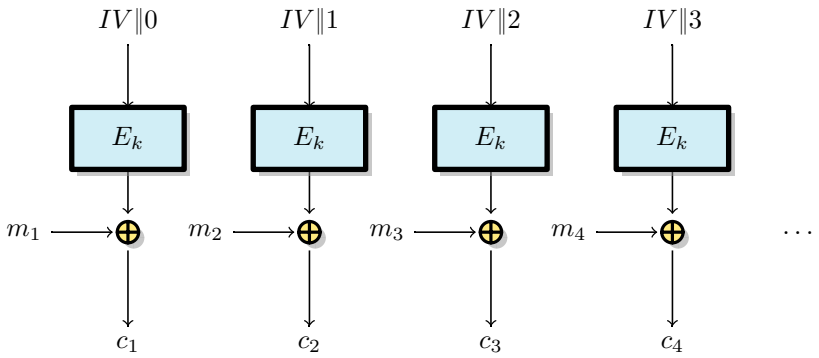
ECB : l'exemple à ne pas suivre. Le mode le plus simple consiste simplement à découper le message par blocs de n bits, puis à chiffrer indépendamment chaque bloc. Ceci est à l'évidence catastrophique (figure 1.4).

CBC : mode séquentiel. Un mode naturel appelé CBC [Ehr+78] consiste à XORer le chiffré d'un bloc au clair du bloc suivant, afin que le bloc suivant dépende du précédent. Pour le premier bloc qui n'a pas de précédent, on XORe une valeur initiale générée aléatoirement (IV). Ce mode peut être attaqué en de l'ordre de $2^{n/2}$ chiffrements. Naturellement, ce mode est purement séquentiel (en chiffrement, mais il peut être parallélisé en déchiffrement).



CTR : mode parallélisé. Le mode compteur [DH79] est très simple, il s'agit de ne pas chiffrer les blocs de message, mais de chiffrer des *nonces* (constantes qui ne doivent être utilisées qu'une seule fois). Ces chiffrés de constantes sont alors utilisés comme une suite chiffrante pseudo-aléatoire et XORés aux blocs de message clair. Typiquement, les constantes peuvent simplement correspondre à un numéro de bloc concaténé à une valeur initiale (IV). Ce mode peut être attaqué en de l'ordre de $2^{n/2}$ chiffrements. Il peut être parallélisé trivialement.

On remarquera que le mode compteur est un chiffrement à flot.



La borne du paradoxe des anniversaires. On remarquera aussi que ces modes sont sécurisés jusqu'à de l'ordre de $2^{n/2}$ chiffrements, même si le chiffrement utilisé à l'intérieur est sécurisé jusqu'à 2^n chiffrements. Ainsi, un chiffrement par blocs dont l'état fait 64 bits pourra être attaqué en de l'ordre de 2^{32} opérations seulement lorsqu'il sera utilisé dans l'un de ces modes d'opérations, ce qui est tout à fait accessible [15] . Un chiffrement dont l'état fait 128 bits pourra être attaqué en de l'ordre de 2^{64} opérations, ce qui moins faible mais encore faisable (à condition d'y mettre les moyens).

Dans le cas des chiffrements à flot, on considère pour cette raison qu'il faut utiliser un état interne de taille au moins deux fois plus grande que la taille de clef.

Cette borne de sécurité découle tout naturellement du paradoxe des anniversaires, qui dit que parmi un calendrier de N dates, la probabilité que parmi $\sqrt{N}$ personnes, au moins deux partagent une date d'anniversaire en commun est non-négligeable.

En particulier, dans un ensemble contenant N éléments, si on tire $\sqrt{N}$ éléments, on s'attend à obtenir une *collision*, c'est-à-dire deux valeurs qui coïncident. Pour $N = 2^n$, on s'attend donc a une collision après $2^{n/2}$ chiffrements (qui donnent des valeurs sensément aléatoires).

Ceci pose problème par exemple dans le mode compteur : lorsqu'on utilise un chiffrement par blocs (qui est donc une famille de permutations), il n'y a *jamais* de collision tant que les entrées sont différentes (en particulier quand les entrées sont un compteur). Donc si on calcule l'image de $2^{n/2}$ blocs (pour une même clef) et qu'il n'y a pas de collision, il y a une forte probabilité que la fonction calculée ne soit pas une fonction aléatoire mais une permutation (ce qui veut dire qu'il sera facile de distinguer la permutation d'une fonction aléatoire).

L'un des objectifs actuels dans la recherche sur les modes opératoires est d'obtenir des modes qui dépassent la borne du paradoxe des anniversaires, *i.e.* qui soient sécurisés après plus de $2^{n/2}$ chiffrements. De tels modes existent (CENC par exemple [Iwa06]), mais sont pour l'heure plus coûteux que les modes cités plus haut. En particulier, ces modes utilisent généralement un *nonce*, une valeur publique mais qui ne doit jamais être réutilisée avec la même clef. Une alternative au nonce est le *tweak*, utilisé notamment dans les chiffrements par blocs « tweakable » [LRW02]. L'utilisation d'un nonce ou d'un tweak apporte néanmoins une contrainte supplémentaire pour l'utilisateur.

1.2.7.2 Chiffrement par blocs itérés

La manière usuelle de construire des chiffrements par blocs consiste à séparer le chiffrement en tours similaires. Chaque tour applique ainsi la même *fonction de tour* avec des *clefs de tour* différentes.

15. Notons tout de même qu'une telle attaque demande 2^{64} données, ce qui est très grand. Ainsi, un état de 64 bits reste relativement sûr en pratique puisqu'on changera généralement de clef avant 2^{64} chiffrements.

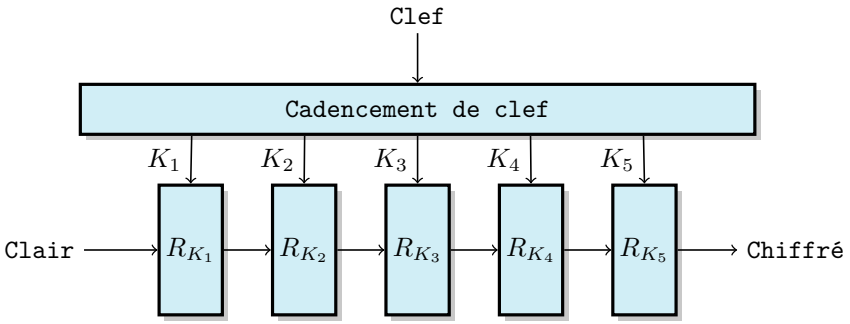


Figure 1.5 – Schéma d'un chiffrement par blocs itéré générique.

Ceci permet d'obtenir des chiffrements complexes en ne construisant qu'une fonction de tour, bien plus simple.

Cadencement de clef. Afin de générer les clefs de tour, on ajoute un cadencement de clef, qui génère toutes les clefs de tour de façon déterministe à partir de la clef de chiffrement. Les propriétés attendues d'un cadencement de clef sont encore mal comprises, mais deux principes essentiels apparaissent : il faut éviter que, pour certains choix de clefs de tour, la sécurité du chiffrement soit dégradée (ce qu'on appelle des *clefs faibles*) et il faut faire en sorte que les clefs de tour se comportent comme des variables aléatoires indépendantes, même si obtenir une telle propriété est très compliqué en général.

1.2.7.3 Les grandes familles de chiffrements par blocs

Il existe deux familles majeures de chiffrements par blocs : les chiffrements de type Feistel, les chiffrements de type SPN.

La recherche sur le sujet est prolifique et le nombre de chiffrements inspirés de ces grandes familles ne cesse d'augmenter. Aucune liste exhaustive de ces chiffrements n'existe, mais on pourra se référer par exemple au site de CryptoLux [16] et en particulier au projet Felics [Din+15], une initiative de l'université de Luxembourg qui regroupe et teste les performances d'un grand nombre de primitives symétriques à bas coût.

Réseau de Feistel. Formalisé par Horst Feistel et son équipe pour la création de Lucifer, puis du Data Encryption Standard (DES 1977), il s'agit d'un chiffrement par blocs itéré dont l'état interne sur n bits (typiquement, $n = 128$) est divisé en deux moitiés de $n/2$ bits. À chaque tour, la moitié de droite est modifiée en y ajoutant (au sens XOR) une transformation non-linéaire de la moitié gauche dépendante de la clef de tour. Puis les deux moitiés sont échangées. On réitère ce procédé autant de fois qu'il est nécessaire pour atteindre la sécurité voulue.

Formellement, si on note R_i la moitié de gauche de l'état au tour i et L_i la moitié de droite, un tour de réseau de Feistel effectue les opérations suivantes :

$$R_i = L_{i-1}\,,$$

$$L_i = R_{i-1} \oplus F_{K_i}(L_{i-1})\,,$$

où F_{K_i} est une fonction non-linéaire paramétrée par la clef de tour K_i.

Les avantages de cette approche sont multiples : (1) on réduit le problème de créer une « bonne » fonction sur n bits au problème de créer une bonne fonction de tour sur $n/2$ bits, (2) il n'est pas nécessaire que la fonction de tour R soit bijective, car l'opération *de type*

16. https://www.cryptolux.org/index.php/Lightweight_Cryptography

Feistel, à savoir XORer à la moitié droite une valeur dépendant de la moitié gauche, est une opération bijective, (3) cette structure est presque involutive,[17] car une opération de type Feistel est involutive. Le déchiffrement consiste donc simplement à effectuer tous les tours dans l'ordre inverse et (4) de manière générale, un schéma de Feistel est relativement peu coûteux pour un chiffrement par blocs.

Quelques exemples de chiffrement de type Feistel sont le DES [Des] (l'ancien standard de chiffrement), Simon et Speck [Bea+13] (les chiffrements à bas coût de la NSA) et LBlock [WZ11].

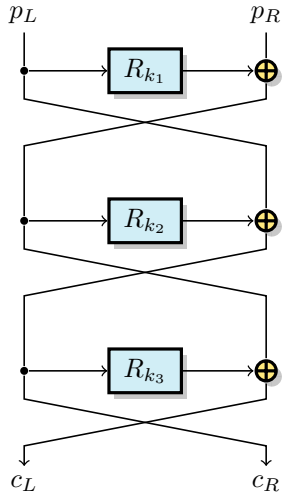


Figure 1.6 – Schéma d'un chiffrement de type Feistel.

Réseau de type MISTY. Formalisé lors de la création du chiffrement MISTY1 par Matsui [Mat97], cette variante des réseaux de Feistel modifie également une moitié de l'état à chaque tour, mais cette modification est légèrement différente du cas des réseaux de Feistel. On applique une opération non-linéaire sur la moitié droite, puis on y XORe la moitié gauche.

Formellement, si on note R_i la moitié de gauche de l'état au tour i et L_i la moitié de droite, un tour de réseau de type MISTY effectue les opérations suivantes :

$$R_i = L_{i-1}\,,$$

$$L_i = F_{K_i}(R_{i-1}) \oplus L_{i-1}\,,$$

où F_{K_i} est une fonction non-linéaire paramétrée par la clef de tour K_i.

Ce schéma partage les avantages du réseau de Feistel, hormis le fait que R doit être bijective. En effet, ici encore, l'inverse du chiffrement consiste simplement à appliquer le réseau de type MISTY en partant de la fin, mais avec pour fonction de tour R^{-1}.

Réseau de type Substitution-Permutation. Les réseaux de Substitution-Permutation (SPN) sont une application directe des principes de Shannon de confusion et de diffusion : un tour est composé

— d'une introduction de la clef de tour par XOR,

— d'une étape non-linéaire faite d'une application en parallèle d'une petite fonction S appelée *boîte-S* sur quelques bits à la fois (typiquement, 4 ou 8 bits), cette étape apportant la confusion,

17. Une involution est une fonction qui est égale à son inverse.

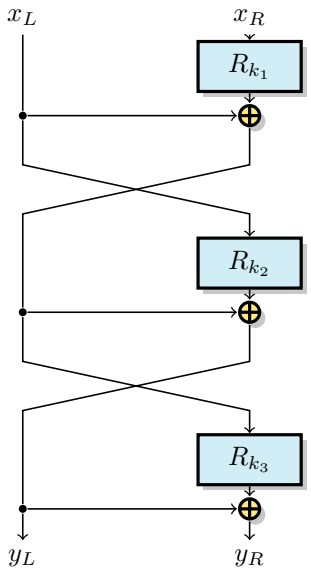


Figure 1.7 – Schéma d'un chiffrement de type MISTY.

— et pour finir d'une étape linéaire qui apporte la diffusion et qui se présente sous la forme d'une grande fonction linéaire sur le bloc entier (typiquement sur 128 bits).

Généralement, les SPN sont plus coûteux que les réseaux de Feistel ou de type MISTY, mais peuvent tout de même atteindre des coûts très faibles. L'avantage majeur de cette structure est qu'un modèle d'analyse baptisé *wide-trail strategy* [DR01] a été développé pour étudier les SPN et permet de justifier de la sécurité d'un SPN sans trop de difficultés.

Ce type de chiffrement a été popularisé par le chiffrement Rijndael, plus tard sélectionné comme AES (Advanced Encryption Standard [DR02]), le standard actuel de chiffrement symétrique. Un grand nombre de SPN existent dans la littérature, entre autres KLEIN [GNL11], Midori [Ban+15], Noekeon [Dae+00], Robin et Fantomas [Gro+15], Skinny [Bei+16], PRESENT [Bog+07] et PRINCE [Bor+12].

Choix des opérations élémentaires et ARX. Lorsqu'il s'agit d'implémenter en pratique les chiffrements sur un ordinateur ou un micro-contrôleur, certaines opérations sont plus simples que d'autres.

Il convient ici de discuter de la structure mathématique des mots de n bits. On considère généralement nos chiffrements comme des permutations agissant sur un ensemble structuré, $\{0, \ldots, 2^n - 1\}$, généralement avec une structure d'espace vectoriel sur le corps fini $\mathbb{F}_{2^n}$, et de ce fait muni d'une addition, d'une multiplication scalaire et d'une inverse.

En revanche, le choix des opérations élémentaires peut varier, en particulier, on considère généralement deux additions distinctes : le XOR bit à bit (sur n bits) et l'addition modulaire dans $\mathbb{Z}/2^n\mathbb{Z}$.

Le choix de l'opération d'addition est essentiel : c'est ce choix qui détermine quelles opérations sont linéaires (et lesquelles ne le sont pas).

Dans la suite de ce document, nous considérerons généralement que l'opération d'addition est un XOR bit à bit dans $\{0, \ldots, 2^n - 1\}$ ce qui est un choix usuel.

L'appellation ARX correspond alors à un choix d'opérations élémentaires : l'Addition modulaire, la Rotation bit à bit et le XOR bit à bit.

Alternativement, plutôt que d'utiliser de telles opérations élémentaires, on peut stocker les images des fonctions en mémoire, et utiliser des accès mémoire pour appliquer la fonction. On considère parfois qu'une boîte-S est une fonction implémentée par sa table

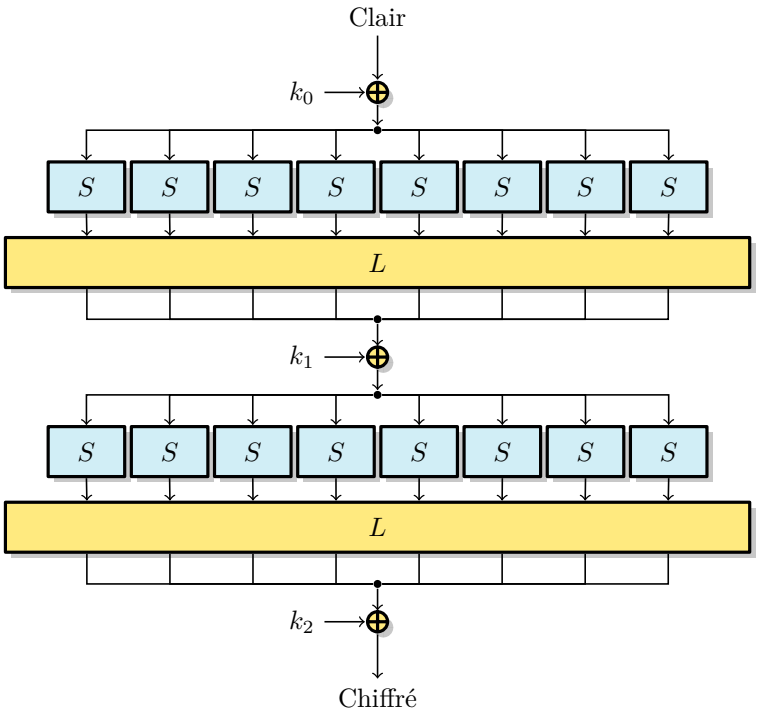


Figure 1.8 – Schéma d'un chiffrement de type SPN (2 tours). Le cadencement de clef est omis pour simplifier.

des images, et on l'oppose alors à un autre type d'implémentation. Dans ce document, nous appellerons boîte-S une fonction sur un (petit) nombre de bits. Son implémentation pourra alors être effectuée à l'aide d'une table des images ou d'une succession d'opérations élémentaires, par exemple.

1.2.8 Le noyau des chiffrements par blocs : la boîte-S

Dans beaucoup de chiffrements par blocs, la partie non-linéaire est le rôle d'un objet particulier appelé boîte-S. Il s'agit ni plus ni moins d'une fonction sur n bits, avec n petit car il est compliqué d'étudier et d'implémenter une fonction sans propriétés particulières pour n grand. Généralement, n est choisi égal à 4 ou 8.

La propriété majeure d'une boîte-S est d'apporter énormément de confusion et de diffusion sur n bits. En ce sens, la boîte-S peut être vue comme un chiffrement miniature (sans clef), sur une taille de fonction où l'étude et l'implémentation sont – relativement – simples.

Remarque 1.1 (Attention !). L'objectif pour un chiffrement *n'est pas* d'apporter le plus possible de confusion et de diffusion. Un chiffrement doit avoir la confusion et la diffusion d'une fonction (bijective) *aléatoire*, c'est-à-dire une confusion et une diffusion *moyennes*. En revanche, pour ce qui est de la boîte-S, plus la boîte-S apporte de confusion, moins on aura besoin de tours pour atteindre une sécurité souhaitée.

La propriété attendue pour une boîte-S est d'être aussi compliquée que possible : un degré algébrique maximal en tant que polynôme dans $\mathbb{F}_{2^n}[X]$, que ses dérivées en tout point prennent le plus de valeurs possibles (optimalement 2^{n-1} valeurs), une transformée de Walsh (et donc une transformée de Fourier discrète) la plus lisse possible, éviter toute propriété invariante, éviter les points fixes et bien d'autres propriétés correspondant à

d'autres types d'attaques.

Il est usuel – mais pas obligatoire – de demander qu'une boîte-S soit bijective (ce qu'on appelle généralement une *permutation*). En particulier, dans le cas des SPN, si la boîte-S n'est pas bijective, le chiffrement ne le sera pas non plus (et donc le déchiffrement ne sera pas bien défini). Cette restriction ne s'applique pas au cas des chiffrements de type Feistel, dont la boîte-S n'est pas appliquée directement sur une branche.

Remarque 1.2. En y regardant de plus près, beaucoup de ces propriétés reviennent à dire qu'une boîte-S doit être aussi loin que possible d'une fonction linéaire (une droite).
En effet, on veut un degré haut pour être aussi loin que possible du degré 1 d'une fonction linéaire.
On veut des dérivées en tout point qui prennent le plus de valeurs possibles pour être aussi loin que possible des dérivées en tout point constantes d'une fonction linéaire.
On veut une transformée de Walsh aussi lisse que possible pour être aussi loin que possible du pic, qui est la transformée d'une fonction linéaire.
On veut éviter toute propriété d'invariance là où une fonction linéaire envoie un groupe sur un autre.
Toutes ces propriétés sont en fait des « mesures » de non-linéarité distinctes, chacune définissant différemment le contraire d'une fonction linéaire.

En plus de ses bonnes propriétés cryptographiques, on souhaite qu'une boîte-S soit simple – et si possible peu coûteuse – à implémenter. A priori, une fonction de n bits peut toujours s'implémenter comme une table stockant toutes les images, comme développé en section 1.2.3, ce qui demande de stocker $n2^n$ bits. Par exemple, pour $n = 8$, l'implémentation d'une boîte-S demande $8 \times 2^8 = 2048$ bits de mémoire, ce qui n'est pas très efficace. En revanche, pour $n = 4$, on n'a à stocker que 64 bits, ce qui est peu coûteux.

Ceci ne résout pas l'un des inconvénients majeurs de l'implémentation par table : la lenteur. En effet, calculer l'image d'une entrée par la boîte-S demande alors un accès mémoire, ce qui est parfois coûteux (suivant la taille de la boîte-S et le contexte de l'implémentation). Néanmoins, une boîte-S de 4 bits peut être implémentée efficacement par une table (par exemple avec des instructions VPERM [Ben+13]), car la table entière tient sur un registre de 64 bits.

Tout comme en section 1.2.3, on est donc ramené à chercher des boîtes-S structurées, donc moins coûteuse, ou à optimiser l'implémentation des boîtes-S existantes. L'étude des boîtes-S, même sur des petites tailles, est déjà complexe, et on voit désormais qu'il n'existe pas de « meilleure » solution, puisqu'il faut faire des compromis. Ainsi le choix de la boîte-S peut dépendre de son utilisation, et on ne peut pas se contenter de chercher une boîte-S « optimale » de façon générique.

Plus de détails sur les boîtes-S sont donnés au chapitre 2

1.2.9 Propagation de la confusion : la fonction de diffusion

Si la boîte-S sert à apporter de la confusion et de la diffusion sur de petits mots de n bits, c'est la vocation de la fonction de diffusion de mélanger ensemble tous ces mots. On estime généralement que la boîte-S est la seule source de confusion et que la fonction de diffusion n'apportera que de la diffusion entre les mots (d'où son nom). Pour cette raison, la fonction de diffusion n'a pas besoin d'être non-linéaire et sera donc choisie linéaire pour simplifier son étude et son implémentation.

Les critères requis pour les matrices de diffusion seront développés au chapitre 3

1.3 Sécurité des chiffrements par blocs : les distingueurs usuels

Rigoureusement, un distingueur est défini pour un chiffrement, *i.e.* un chiffrement par blocs utilisé dans un mode opératoire[18]. En revanche, étudier les façons de distinguer un chiffrement par blocs d'une PRP de taille fixe (ou d'une PRF de taille fixe) peut donner des informations sur la sécurité du chiffrement.

Remarque 1.3. On remarquera que les modèles d'attaques (*i.e.* attaque à clair connu/choisi, etc) n'ont a priori de sens que sur un chiffrement, pas sur un chiffrement par blocs. Par abus de langage, on assimilera parfois l'entrée d'un chiffrement par blocs à un clair et sa sortie à un chiffré (bien que cela n'aie pas de sens en général, le mode compteur par exemple ne chiffre qu'un compteur (qui peut être connu, mais pas choisi par l'attaquante)).

Ainsi, on pourra s'intéresser à distinguer un chiffrement par blocs d'une PRP de taille fixe ou d'une PRF de taille fixe (cf. Section 1.2.2). Dans ce cadre, les trois types d'attaques classiques consistent à mesurer une certaine « distance » par rapport aux fonctions linéaires. On mesurera d'une part cette distance pour une permutation à clef aléatoire tirée dans le chiffrement par blocs (notons cette distance d_E) et pour une PRP (notons cette distance $d_\$$). Alors si $|d_E - d_\$|$ est grande, on sera capable de distinguer le chiffrement par blocs d'une PRP.

Les trois attaques classiques correspondent à trois mesures différentes de « linéarité » (*i.e.* de distance à l'espace des fonctions linéaires). Pour ces trois mesures de linéarité, des études existent qui mesurent la distance entre une PRP (ou une PRF) et une fonction linéaire. Dit simplement, une PRP (ou une PRF) est loin de toute fonction linéaire pour chacune de ces trois mesures de linéarité. Ainsi, il faudra qu'un permutation aléatoire dans le chiffrement par blocs soit loin de toute fonction linéaire.

Le *distingueur algébrique* consiste à étudier le degré d'une permutation à clef (sachant que le degré d'une fonction linéaire est 1 par définition). Si le degré d'une permutation aléatoire du chiffrement par blocs est trop bas, on pourra distinguer le chiffrement par blocs d'une PRP.

Le *distingueur différentiel* consiste à étudier la dérivée d'une permutation à clef (sachant que, pour une fonction linéaire, ses dérivées en tout point sont constantes). Si les dérivées d'une permutation aléatoire du chiffrement par blocs ne prennent pas beaucoup de valeurs, on pourra distinguer le chiffrement par blocs d'une PRP.

Le *distingueur linéaire* consiste à étudier la distance de Hamming entre une permutation à clef et n'importe quelle fonction linéaire. Si cette distance est trop faible, on pourra distinguer le chiffrement par blocs d'une PRP.

1.3.1 Un exemple d'attaque par distingueur : l'attaque sur le dernier tour

Avant de rentrer dans les détails des distingueurs usuels, voyons comment un distingueur sur un chiffrement par blocs (sans mode opératoire) peut permettre de retrouver de l'information sur la clef ou sur des messages clairs. Un exemple d'une telle transformation distingueur → attaque est l'attaque sur le dernier tour.

Le principe est le suivant : soit $(E_k)_{k\in\mathbb{F}_2^K}$ un chiffrement par blocs itéré de r tours avec une taille de blocs n. Une attaque sur le dernier tour permet de retrouver la clef (ou de l'information) dès qu'on connaît une propriété permettant de différencier l'application des $(r-1)$ premiers tours d'une permutation aléatoire.

18. Rappelons qu'un chiffrement par blocs est une famille de permutations (à clef) de taille d'entrée fixée, là où un chiffrement est une famille de permutations (à clef) de taille d'entrée variable

L'idée est la suivante : une attaquante ne connaît pas la clef, mais elle connaît une propriété probabiliste permettant de distinguer les $(r-1)$ premiers tours d'une PRP. Elle va tester itérativement toutes les valeurs de la sous-clef du dernier tour (on supposera que la sous-clef du dernier tour est plus petite que la clef-maître). On appellera k_r la sous-clef du dernier tour et $\tilde{k_r}$ la valeur de sous-clef du dernier tour que teste l'attaquante. On suppose que l'attaquante connaît un grand nombre de couples (clair, chiffré). Pour chacun des textes chiffrés, elle va inverser uniquement le dernier tour en utilisant sa clef $\tilde{k}$. Si $k \neq \tilde{k}$, le résultat de cette inversion aura des propriétés aléatoires. En revanche, si $k = \tilde{k}$, alors le résultat de l'inversion correspondra à la propriété distingante des $(r-1)$ premiers tours du chiffrement par blocs. Ainsi, si la propriété attendue est vérifiée, c'est que $k = \tilde{k}$ donc elle a trouvé la clef. Elle réitère ce procédé jusqu'à tomber sur la bonne clef, c'est-à-dire celle qui vérifie la propriété après inversion du dernier tour.

Ainsi, on voit que, dès qu'on connaît une propriété distingante sur le chiffrement par blocs, on est capable de retrouver la clef. Pour concevoir un chiffrement par blocs sûr, il faut donc faire en sorte que tester une propriété distingante soit trop coûteux pour l'adversaire, c'est-à-dire que le chiffrement ne puisse pas être distingué d'une permutation aléatoire avec des moyens raisonnables. Notons que, même si un distingueur ne permet pas toujours de retrouver la clef, il permet souvent de retrouver de l'information sur le message clair.

1.3.2 Les attaques différentielles

Les attaques différentielles ont été introduites par Biham et Shamir en 1991 [BS91]. L'idée est d'étudier les propriétés de la différentielle du chiffrement et de comparer ces propriétés à celles attendues pour une fonction aléatoire.

Remarque 1.4. Notons ici un choix important. L'objectif étant de rendre impraticable un distingueur différentiel, le critère de sécurité rigoureux est le suivant : un chiffrement par blocs $E = (E_k)_{k \in \mathbb{F}_2^K}$ est indistingable différentiellement d'une PRP si une permutation aléatoire $E_k \in E$ vérifie que la distribution de sa différentielle est proche de la distribution de la différentielle d'une PRP.

L'inconvénient, c'est qu'étudier la distribution de la différentielle d'une permutation d'un grand nombre de bits (typiquement 128) est très compliqué. C'est pourquoi on limite notre étude à une sous-distribution : le maximum de la distribution de la différentielle.

Si cette étude du maximum ne peut pas suffire à prouver la résistance face aux distingueurs différentiels, elle suffit néanmoins en pratique, car les attaques différentielles efficaces en pratique exploitent toutes ce maximum.

Ainsi, pour une fonction F (par exemple $F = E_k$), on considère deux messages en entrée x et $x + a$, et on regarde la différence entre les sorties $F(x)$ et $F(x + a)$. Si l'on peut dire qu'avec forte probabilité, une différence en entrée a implique une différence en sortie b, ceci permet de distinguer F d'une fonction aléatoire.

1.3.2.1 Uniformité différentielle et table des différences

Formellement, une attaque différentielle est définie entre deux groupes $(A, \oplus)$ et $(B, \boxplus)$ [19]. Elle consiste à étudier, pour $F : A \to B$, la distribution de la *dérivée* (ou *différentielle*) de F en n'importe quel point a, définie par :

$$\begin{aligned} D_a F : A &\to B \\ x &\mapsto F(x) \boxminus F(x \oplus a), \end{aligned}$$

19. Généralement, on choisit $A = B$ et $\oplus$ et $\boxplus$ le XOR bit à bit.

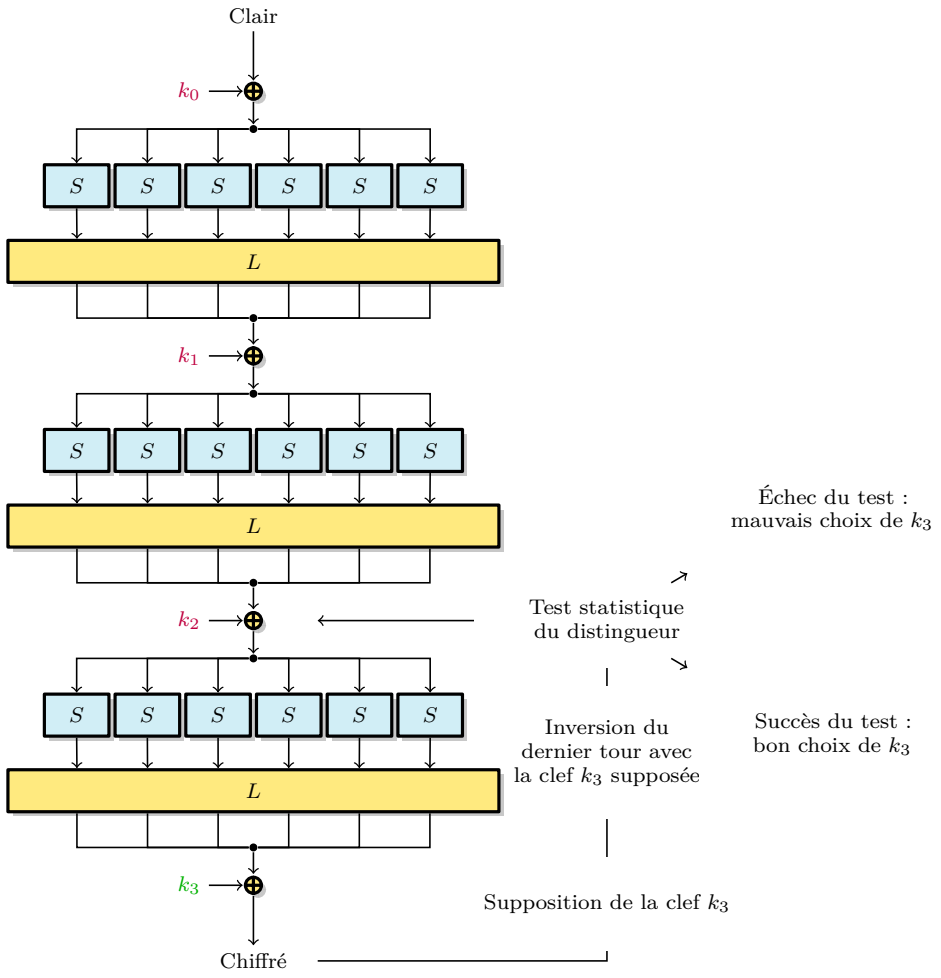


Figure 1.9 – Principe d'une attaque sur le dernier tour (exemple d'un SPN de 3 tours). Les clefs violettes sont inconnues de l'attaquante, la clef verte est une supposition de l'attaquante.

où $\boxminus$ est l'opposé de $\boxplus$.

Si l'on peut identifier une différence en entrée a et une différence en sortie b telles qu'avec forte probabilité, une différence de a sur l'entrée entraîne une différence de b sur la sortie, on a un distingueur.

Formellement, avec

$$\delta_F(a,b) = \#\{x \in A | D_a F(x) = b\},$$

s'il existe $a \in A^*$ (*i.e.* $A\backslash\{0\}$) et $b \in B$ tels que $\delta_F(a,b)$ est grand [20], ceci donne un distingueur. Plus $\delta_F(a,b)$ est grand, plus le distingueur est efficace (i.e. moins il sera coûteux de distinguer F d'une fonction aléatoire).

L'objectif pour un concepteur sera donc que la distribution de $\delta_F(a,b)$ se rapproche de la distribution attendue pour la dérivée d'une fonction aléatoire.

Le distingueur différentiel. L'idée du distingueur sera de choisir les différences a et b qui maximisent $\delta_F(a,b)$, puis de tester si $D_a F(x) = b$ pour de nombreux x. Si beaucoup de x

20. Cette notion de « grand » dépend du choix de groupe, mais en général, on pourra considérer grand comme $\gg \#A/\#B$.

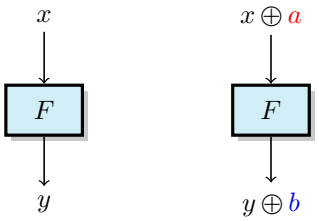


Figure 1.10 – Principe d'une attaque différentielle (les variables en couleur sont des différences).

vérifient $D_aF(x) = b$, on dira qu'il s'agit de F, sinon on dira qu'il s'agit d'une permutation aléatoire.
Dans le cas usuel où les groupes en entrée et en sortie de F sont $(\{0,1\}^n, \mathsf{XOR})$, il faudra de l'ordre de $\frac{2^{n+1}}{\delta_F(a,b)}$ couples (entrée, sortie) pour distingueur F d'une permutation aléatoire, avec a et b maximisant $\delta_F(a,b)$.

Définition 1.1 (Table des différences)**.** Soient deux groupes $(A,\oplus)$ et $(B,\boxplus)$, et $F : A \to B$. On appelle *table des différences* de F et on note $\mathsf{DDT}(F)$ la table contenant en ligne a, colonne b, la valeur $\delta_F(a,b)$, pour tous $a \in A$ et $b \in B$.

On représente généralement la DDT par un tableau à deux dimensions, avec les différences en entrée (dans A) sur les lignes et les différences en sortie (dans B) sur les colonnes.

Définition 1.2 (Uniformité différentielle)**.** Soient deux groupes $(A,\oplus)$ et $(B,\boxplus)$, et $f : A \to B$. On appelle *uniformité différentielle* de f et on note $\delta(f)$ le maximum de $\mathsf{DDT}(f)$ privée de la première ligne (cas $a = 0$), *i.e.* :

$$\delta(f) = \max_{a \neq 0, b} \#\{x \in A | f(x) \boxminus f(x \oplus a) = b\}.$$

On utilisera souvent une normalisation de l'uniformité différentielle, appelée *probabilité différentielle*, et qui vaut $\frac{\delta(f)}{\#A}$.

Propriété 1.1 (Parité de $\delta_F(a,b)$)**.** *Soient deux groupes $(A,\oplus)$ et $(B,\boxplus)$. Si $\oplus$ et $\boxplus$ sont telles que pour tout $a \in A$, $a \oplus a = 0$ et pour tout $b \in B$, $b \boxplus b = 0$, et que $\boxplus$ est commutative, alors quel que soit $a \in A^*$ et quel que soit $b \in B$, $\delta_F(a,b)$ est pair.*

Démonstration. En effet, pour tout $a \in A^*$, pour tout $b \in B$, si $x \in A$ est solution de $F(x) \boxminus F(x \oplus a) = b$, par involutivité, on a $F((x \oplus a) \oplus a) \boxminus F(x \oplus a) = b$ et par commutativité, $F(x \oplus a) \boxplus F((x \oplus a) \oplus a) = b$, donc si x est solution de $D_aF(x) = b$, alors $x \oplus a$ l'est aussi. Donc $\{x \in A | D_aF(x) = b\}$ est formé de paires. □

Remarque 1.5. On remarquera que non seulement $\delta_F(a,b)$ est toujours pair pour $a \neq 0$, mais en plus, les solutions de toute équation différentielle de la forme $D_aF(x) = b$ vont par paires.

En particulier, ceci est vérifié dans le cas usuel où les groupes $(A,\oplus)$ et $(B,\boxplus)$ sont égaux à $(\{0,1\}^n, \mathsf{XOR})$.

L'uniformité différentielle comme mesure de linéarité. Lorsque les fonctions linéaires sont définies (c'est-à-dire si on travaille sur des espaces vectoriels plutôt que des groupes), l'uniformité différentielle peut être interprétée comme une mesure de linéarité : il s'agit d'étudier la particularité de la dérivée en tout point d'une fonction linéaire.
La dérivée en tout point a d'une fonction linéaire est constante, *i.e.*, pour tout $a \in A$, il

existe $\in B$ tel que $D_a F(x) = b$ pour tout x. Ainsi, si l'uniformité différentielle de F vaut $\#A$, c'est que l'une des dérivées de F est constante, ce qui veut dire que F est proche d'une fonction linéaire. Plus $\delta(F)$ est petite, plus F est loin d'une fonction linéaire.

1.3.2.2 MEDP : le critère de résistance aux attaques différentielles pour un chiffrement par blocs

L'uniformité différentielle et la table des différences permettent d'étudier la dérivée d'une fonction. Or, nous voulons étudier la dérivée d'une permutation à clef aléatoire d'un chiffrement par blocs $(E_k)_{k\in\mathbb{F}_2^K}$ à valeurs dans $\mathbb{F}_2^n$. Naturellement, on définit alors le MEDP(Maximum Expected Differential Probability), qui correspond au maximum de l'espérance des probabilités différentielles (où l'espérance est prise sur le choix de clef) :

$$\mathsf{MEDP}((E_k)_{k\in\mathbb{F}_2^K}) = \max_{a\in(\mathbb{F}_2^n)^*, b\in\mathbb{F}_2^n} \mathsf{Moy}_{k\in F_2^K} \frac{\delta(E_k)}{2^n}\,.$$

Ceci correspond en quelque sorte à l'uniformité différentielle d'une permutation à clef « moyenne » dans $(E_k)_{k\in\mathbb{F}_2^K}$.

1.3.3 Les attaques linéaires

Le principe des attaques linéaires a été initialement introduit par Gilbert, Chassé et Tardy-Corfdir [GC91 ; TCG92] puis appliquées par Matsui [Mat94a ; Mat94b] pour attaquer DES.

L'idée majeure de ce type d'attaque est d'approcher la fonction F par une fonction linéaire, ce qui revient à trouver une relation linéaire biaisée entre l'entrée et la sortie de F, *i.e.* trouver deux valeurs a et b non-nulles telles que la distribution de $x \mapsto a \cdot F(x) \oplus b \cdot x$ ne soit pas uniforme.

Tout comme le cas différentiel, on n'est pas capable d'étudier la distribution entière de $x \mapsto a \cdot F(x) \oplus b \cdot x$, et on se contente du maximum de cette distribution.

Je ne rentrerai pas ici dans autant de détails que sur les attaques différentielles, mais le fonctionnement est aussi bien plus général que le cas usuel. Je me contenterai ici de considérer des espaces vectoriels usuels et je ne rentrerai pas dans les détails des preuves.

Formellement, on définit une attaque linéaire entre deux espaces vectoriels. Considérons simplement le corps $A = (\mathbb{F}_2, \mathsf{xor}, \mathsf{and})$ et le A-espace vectoriel $(\mathbb{F}_2^n, \mathsf{XOR}, .)$, où . est une multiplication scalaire telle que pour $x \in \mathbb{F}_2^n$, $b \in \mathbb{F}_2$ $x.b = 0$ si $b = 0$ et $x.b = x$ si $b = 1$ [21]. On munit cet espace vectoriel du produit scalaire $\cdot$ défini par $x \cdot y = \sum_{1\leqslant i\leqslant n} x_i \text{ and } y_i$, où la somme considère une addition comme un xor binaire, x_i est le i-ième bit de x.

Alors si

$$\mathbb{P}(a \cdot F(X) \oplus b \cdot X = 1) = \tfrac{1}{2} + \varepsilon \text{ avec } \varepsilon \in [-\tfrac{1}{2}, \tfrac{1}{2}]\,,$$

il est possible de distinguer F d'une fonction aléatoire en approximativement ε^{-2} calculs de F. Ainsi, si $|\varepsilon| \gg 2^{-\frac{n}{2}}$, on aura un distingueur en moins de 2^n calculs.

Définition 1.3 (Transformée de Walsh)**.**
Soit f une fonction de $(\mathbb{F}_2^n, \mathsf{XOR}, .)$ dans $(\mathbb{F}_2, \mathsf{xor}, \mathsf{and})$ (*i.e.* une fonction booléenne) [22]. On appelle *transformée de Walsh* de f (ou parfois transformée de Walsh-Hadamard ou encore transformée de Fourier discrète) et on note $\hat{f}$ la fonction définie par

$$\hat{f}(a) = \sum_{x\in\mathbb{F}_2^n} (-1)^{f(x)\oplus a\cdot x}\,.$$

21. On se sert très peu de la multiplication scalaire.
22. Dans la suite, on notera indifféremment XOR et xor par le symbole $\oplus$, qui représentera un xor binaire s'il est défini sur $\mathbb{F}_2$ ou un XOR bit à bit sur n bits s'il est de $\mathbb{F}_2^n$ dans $\mathbb{F}_2^n$.

On appelle $\hat{f}$ le coefficient de Walsh de f au point $a \in \mathbb{F}_2^n$.

Soit F une fonction de $(\mathbb{F}_2^n, \oplus, .)$ dans $(\mathbb{F}_2^n, \oplus, .)$. On appelle *transformée de Walsh* de F la fonction $\hat{F}$ définie par

$$\hat{F}(a,b) = \hat{F}_b(a) \sum_{x \in \mathbb{F}_2^n} (-1)^{b \cdot F(x) \oplus a \cdot x},$$

où $F_b : x \mapsto b \cdot F(x)$ sont appelées les *composantes* de F.

En y regardant de près, un coefficient de Walsh $\hat{F}(a,b)$ compte la proportion des $x \in \mathbb{F}_2^n$ qui vérifient $b \cdot F(x) \oplus a \cdot x = 0$, c'est-à-dire des x pour lesquels $b \cdot F(x)$ est égale à la fonction linéaire $a \cdot x$. Chaque coefficient de Walsh mesure donc une distance de F à une fonction linéaire de $\mathbb{F}_2^n$ dans $\mathbb{F}_2^n$.

Précisément, $\hat{F}(a,b) = 2^n - 2d_H(a \cdot F(x), b \cdot x)$, où d_H est la distance de Hamming, donc $\hat{F}(a,b)$ est une mesure de la distance entre $a \cdot F$ et $b \cdot x$.

L'objectif étant que F soit le plus loin possible de toute fonction linéaire, on cherche donc f telle que pour tous a et b non-nuls, $\hat{F}(a,b)$ est le plus petit possible. Tout comme le cas des attaques différentielles, on représente généralement les coefficients de Walsh $\hat{F}(a,b)$ dans un tableau à deux dimensions dont les lignes correspondent au masque linéaire d'entrée a et les colonnes au masque linéaire de sortie b, et on s'intéressera à minimiser le maximum des coefficients de Walsh $\hat{F}(a,b)$ pour la boîte-S S.

Définition 1.4 (Table des biais linéaires)**.** Soit F une fonction de $(\mathbb{F}_2^n, \oplus, .)$ dans $(\mathbb{F}_2^n, \oplus, .)$. On appelle *table des biais linéaires* de F et on note $\mathsf{LAT}(F)$ la table à deux dimensions contenant en ligne a, colonne b, la valeur $\hat{F}(a,b)$, pour tous $a \in \mathbb{F}_2^n$, $b \in \mathbb{F}_2^n$.

Définition 1.5 (Linéarité)**.** Soit F une fonction de $(\mathbb{F}_2^n, \oplus, .)$ dans $(\mathbb{F}_2^n, \oplus, .)$. On appelle *linéarité* de F et on note $\mathcal{L}(F)$ le maximum (en valeur absolue) de la LAT de S, i.e.[23]

$$\mathcal{L}(F) = \max_{a \neq 0, b} |\hat{F}(a,b)|.$$

On considérera parfois de manière équivalente la non-linéarité de F, définie par

$$\mathcal{NL}(F) = 2^{n-1} - \tfrac{1}{2}\mathcal{L}(F)$$

car on a l'identité :

$$\mathcal{NL}(F) = \min_{a \in (\mathbb{F}_2^n)^*, b \in \mathbb{F}_2^n, \varepsilon \in \mathbb{F}_2} d_H(a \cdot F(x), b \cdot x + e).$$

La non-linéarité correspond donc à la plus petite distance de Hamming entre F et une fonction affine.

1.3.3.1 MELP : le critère de résistance aux attaques linéaires pour un chiffrement par blocs

Tout ici se passe similairement au cas différentiel : la linéarité et la table des biais linéaires permettent d'étudier la distance de Hamming entre une fonction et l'espace des fonctions affines. Or, nous voulons étudier cette distance de Hamming pour une permutation à clef aléatoire d'un chiffrement par blocs $(E_k)_{k \in \mathbb{F}_2^K}$ à valeurs dans $\mathbb{F}_2^n$. Naturellement, on définit alors le MELP(Maximum Expected Linear Potential), qui correspond au maximum de l'espérance des biais linéaires (où l'espérance est prise sur le choix de clef) :

$$\mathsf{MELP}((E_k)_{k \in \mathbb{F}_2^K}) = \max_{a \in (\mathbb{F}_2^n)^*, b \in \mathbb{F}_2^n} \mathsf{Moy}_{k \in F_2^K} \frac{\mathcal{L}(E_k)}{2^n}.$$

Ceci correspond en quelque sorte à la linéarité d'une permutation à clef « moyenne » dans $(E_k)_{k \in \mathbb{F}_2^K}$.

23. Notons que, si on appellera par la suite $\mathcal{L}(F)$ la linéarité de F, ce n'est en fait que l'une des mesures de linéarité que l'on utilise, puisque $\delta(F)$ par exemple en est une autre.

1.3.4 Wide-trail strategy : réduction de la sécurité des chiffrements par blocs à des critères sur les boîtes-S

Pour analyser des chiffrements par blocs sur des tailles aussi grandes que 128 bits, il est nécessaire d'avoir un *modèle d'analyse* qui puisse réduire la sécurité du chiffrement à des critères sur ses composantes.

C'est ce que fait la *wide-trail strategy*. En particulier, pour les boîtes-S, c'est ce modèle d'analyse qui fait le lien entre l'uniformité différentielle δ d'une boîte-S ainsi que sa linéarité $\mathcal{L}$ et la résistance du chiffrement par blocs aux attaques différentielles et linéaires. De même, c'est la wide-trail strategy qui définit les critères de sécurité pour les matrices de diffusion.

Le modèle d'analyse dépend des schémas, mais le plus poussé est celui des SPN, baptisé « wide-trail strategy » (stratégie du chemin large), introduite lors de la construction du standard AES [DR02].

L'intérêt principal de la wide-trail strategy est qu'elle offre un résultat simple. L'idée est de compter le nombre de boîtes-S « actives » dans le chiffrement par blocs, où la définition d'active dépend de l'attaque considérée.

Prenons ici l'exemple des attaques différentielles. Tout ce qui suit est défini de façon similaire pour les attaques linéaires, mais on ne détaillera que les attaques différentielles par souci de simplicité.

Définition 1.6 (Caractéristique différentielle)**.** Soit F une fonction de $\mathbb{F}_2^n$ dans $\mathbb{F}_2^n$, décomposée en r tours R_i, $1 \leqslant i \leqslant r$. Rappelons qu'on appelle différentielle (a, b) à travers F la donnée d'une différence en entrée de F a et d'une différence en sortie b, et qu'on s'intéresse au nombre de solutions de toute différentielle (*i.e.* le nombre de $x \in \mathbb{F}_2^n$ tels que $F(x) \oplus F(x \oplus a) = b$).

On appelle *caractéristique différentielle* la donnée de $r+1$ différences d_i, $1 \leqslant i \leqslant r+1$ et le *nombre de solutions d'une caractéristique différentielle* le nombre de $x \in \mathbb{F}_2^n$ tels que pour tout $j \in \{0, \ldots, r\}$, $R_j \circ R_{j-1} \circ \ldots R_0(x)$ soit une solution de la différentielle $(d_j, d_j + 1)$ à travers R_{j+1}. Concrètement, une différentielle définit la différence d'entrée de F et la différence de sortie, tandis qu'une caractéristique différentielle définit les différences d'entrée et de sortie de tous les tours R_i.

À l'évidence, on peut décomposer une différentielle en caractéristique : notons $\delta_F^{\mathsf{Car}}(d_0, \ldots, d_{r+1})$ le nombre de solutions d'une caractéristique différentielle $(d_i)_{1 \leqslant i \leqslant r+1}$, alors

$$\delta_F(d_1, d_{r+1}) = \sum_{(d_2, d_3, \ldots, d_r) \in (\mathbb{F}_2^n)^{r-1}} \delta_F^{\mathsf{Car}}(d_1, \ldots, d_{r+1}) \,.$$

On appelle *probabilité d'une caractéristique différentielle* $(d_i)_{1 \leqslant i \leqslant r+1}$ le nombre de solutions normalisé :

$$\mathbb{P}((d_i)_{1 \leqslant i \leqslant r+1}) = \frac{\delta_F^{\mathsf{Car}}(d_1, \ldots, d_{r+1})}{2^n} \,.$$

1.3.4.1 Réduction à un tour moyen

Une première approche de la wide-trail strategy permet, sous une hypothèse forte, de réduire la probabilité d'une caractéristique différentielle à la probabilité différentielle des tours R_i, *i.e.* $\mathbb{P}_{R_i}(u, v) = \frac{\delta_{R_i}(u,v)}{2^n}$.

Proposition 1.1 (Probabilité d'une caractéristiques)**.** *Sous hypothèse que tous les tours sont indépendants (au sens probabiliste) deux à deux :*

$$\mathbb{P}((d_i)_{1 \leqslant i \leqslant r+1}) = \prod_{1 \leqslant i \leqslant r} \mathbb{P}_{R_i}(d_i, d_{i+1}) \,.$$

Dans le cas d'un chiffrement par blocs itéré, cette hypothèse implique en particulier que les sous-clefs de tours doivent se comporter comme des variables aléatoires indépendantes (ce qui n'est jamais le cas dans les SPN utilisés en pratique). Ce modèle d'analysé est donc approximatif.

Admettons que l'hypothèse d'indépendance soit vérifiée. Il ne reste alors qu'à calculer la probabilité de chaque tour indépendamment.

L'idée est alors de considérer un tour R moyen, c'est-à-dire dont la probabilité différentielle est la moyenne des probabilités différentielles des tours R_i. On fera alors l'approximation que

$$\mathbb{P}((d_i)_{1\leqslant i\leqslant r+1}) = \mathbb{P}(R)^r\,.$$

1.3.4.2 Réduction au nombre de boîtes-S actives.

Une approche plus précise est de compter le nombre de boîtes-S *actives* dans un chiffrement par blocs. Dans le cas différentiel, on définit ainsi une boîte-S active :

Définition 1.7 (Boîte-S active)**.** Soit $(E_k)_{k\in\mathbb{F}_2^K}$ un chiffrement par blocs suivant une structure de SPN en r tours avec une taille de bloc N. Nous supposerons (par simplicité) que l'étage de confusion consiste en l'application en parallèle d'une même boîte-S S sur n bits, et ce pour tous les tours, et que la matrice de diffusion M de dimension $m \times m$ sur des mots de n bits est la même pour tous les tours. On négligera la manière dont sont générées les clefs de tour k_i.

Soit $k \in \mathbb{F}_2^K$. Soit $(d_1, \ldots, d_{r+1})$ une caractéristique différentielle à travers E_k. On appelle *boîte-S active* pour la caractéristique différentielle $(d_1, \ldots, d_{r+1})$ toute boîte-S dont la différence en entrée est non-nulle. Par opposition, on appelle *boîte-S inactive* toute boîte-S dont la différence en entrée est nulle.

Les boîtes-S inactives pour une caractéristique différentielle ont donc une différence en entrée nulle et avec probabilité 1 une différence en sortie nulle. Seules les boîtes-S actives influent donc sur la probabilité d'une caractéristique différentielle.

Hypothèse : Les tours R_i sont indépendants deux à deux (*i.e.* $\forall i, j, \mathbb{P}(R_i \text{ et } R_j) = \mathbb{P}(R_i)\mathbb{P}(R_j)$),

Dans ce cas, en notant $nb_{\mathsf{actives}}(d_1, \ldots, d_{r+1})$ le nombre de boîtes-S actives pour la caractéristique $(d_1, \ldots, d_{r+1})$ à travers E_k, on a :

$$\mathbb{P}((d_i)_{1\leqslant i\leqslant r+1}) \leqslant \delta(S)^{nb_{\mathsf{actives}}(d_1,\ldots,d_{r+1})}\,.$$

Il ne reste plus qu'à estimer $nb_{\mathsf{actives}}(d_1, \ldots, d_{r+1})$ pour toute caractéristique différentielle.

Le nombre de boîtes-S actives est une propriété de diffusion, qui dépend uniquement de la matrice de diffusion M. On commencera par estimer le nombre de boîtes-S actives pour toute caractéristique à travers 2 tours de E_k. Ceci est mesuré par le *facteur de branchement différentiel* de M :

Définition 1.8 (Facteur de branchement différentiel)**.** Soit M une matrice de dimension $m \times m$ sur des mots dans $\mathbb{F}_2^n$, *i.e.* $M \in M_k\big(M_n(\mathbb{F}_2)\big)$. On appelle facteur de branchement différentiel noté $\mathcal{B}_d(M)$ la valeur

$$\mathcal{B}_d(M) = \min_{x\neq 0}\{w(x) + w(M(x))\}\,.$$

Le facteur de branchement différentiel considère qu'il y a une différence (non-nulle) en entrée de M, et considère $w(x) + w(M(x))$, c'est-à-dire le nombre de mots avec une différence en entrée de M plus le nombre de mots avec une différence en sortie de M, et

prend le minimum de ces valeurs pour tout $x \neq 0$. Ceci garantit que s'il y a une différence en entrée de M, il y aura au moins $\mathcal{B}_d(M)$ mots avec une différence non-nulle parmi les mots d'entrée et de sortie de M.

En particulier, si M est utilisée comme matrice de diffusion au tour i, cela signifie que s'il y a une différence en entrée de M au tour i, il y aura au moins $\mathcal{B}_d(M)$ boîtes-S actives parmi celles des tours i et $i+1$.

On décompose alors un SPN par paire de tours, et on obtient

$$nb_{\mathsf{actives}}(d_1, \ldots, d_{r+1}) \simeq \mathcal{B}_d(M) \times \tfrac{r}{2} \,.$$

pour toute caractéristique différentielle $(d_1, \ldots, d_{r+1})$, ce qui implique sous hypothèse d'indépendance des tours :

$$\mathbb{P}((d_i)_{1 \leqslant i \leqslant r+1}) \leqslant \delta(S)^{\mathcal{B}_d(M) \times \frac{r}{2}} \,.$$

Ainsi, on ramène la probabilité de toute caractéristique différentielle à l'uniformité différentielle de S et au facteur de branchement différentiel de M.

1.3.4.3 Définitions pour les attaques linéaires

Similairement, on définit pour les attaques linéaires un *chemin linéaire* qui correspond à la donnée des masques linéaires $(a_1, \ldots, a_{r+1})$, on peut décomposer tout coefficient de Walsh $\hat{F}(a_1, a_{r+1}$ en chemins linéaires à travers F. On définit une boîte-S active pour un chemin linéaire comme une boîte-S dont le masque linéaire d'entrée est non-nul, et on compte le nombre de boîtes-S actives pour estimer le potentiel d'une approximation linéaire.

Finalement, sous des hypothèses d'indépendance des tours similaires, on obtient qu'on peut estimer le potentiel d'une approximation linéaire avec la linéarité de S et le facteur de branchement linéaire de M défini par :

Définition 1.9 (Facteur de branchement linéaire)**.** Soit M une matrice de dimension $m \times m$ sur des mots dans $\mathbb{F}_2^n$, *i.e.* $M \in M_k\big(M_n(\mathbb{F}_2)\big)$. On appelle facteur de branchement linéaire noté $\mathcal{B}_l(M)$ la valeur

$$\mathcal{B}_l(M) = \min_{x \neq 0} \{w(x) + w(M^\intercal(x))\} \,,$$

où $M^\intercal$ est la transposée de M.

1.3.5 Matrices de diffusion MDS

Ainsi, les critères à étudier pour la résistance aux attaques différentielles et linéaires pour les matrices de diffusion sont respectivement les facteurs de branchement différentiel et linéaire.

Les bornes optimales pour ces critères pour une matrice de dimension $m \times m$ sont connus grâce à la théorie des codes correcteurs d'erreurs :

$$\mathcal{B}_d(M) \leqslant m+1$$

$$\mathcal{B}_l(M) \leqslant m+1 \,.$$

En particulier, on sait que si $\mathcal{B}_d(M) = m+1$, alors $\mathcal{B}_l(M) = m+1$ et réciproquement . De plus, on sait construire des matrices M atteignant ce maximum en utilisant des codes MDS. C'est pourquoi, par analogie, on appelle une matrice M telle que $\mathcal{B}_d(M) = \mathcal{B}_l(M) = m+1$ une *matrice MDS*.

Les matrices MDS sont donc optimales en termes de diffusion différentielle et linéaire d'après la wide-trail strategy.

On définira aussi les matrices quasi-MDS M telles que $\mathcal{B}_d(M) = \mathcal{B}_l(M) = m$.

Puisque l'on sait construire des matrices MDS (en particulier la matrice MixColumns d'AES), le problème est maintenant de trouver les matrices MDS les plus efficaces, c'est-à-dire les moins coûteuses. Ce problème est abordé au chapitre 3.

1.3.6 Conditions sur les boîtes-S

Définition 1.10 (Boîte-S)**.** On appelle *boîte-S* une fonction booléenne vectorielle $\mathbb{F}_2^n \to \mathbb{F}_2^m$, ou de manière équivalente une fonction $\mathbb{F}_{2^n} \to \mathbb{F}_{2^m}$.

Même sans la wide-trail strategy, on est capable de définir les bonnes propriétés à avoir pour une boîte-S.

Remarque 1.6. Condition sur une boîte-S Sur un chiffrement par blocs construit à partir d'une boîte-S S, on sait que la boîte-S apporte toute la confusion du chiffrement par blocs, et en particulier toute sa non-linéarité. Il faut donc que S soit le plus non-linéaire possible, c'est-à-dire que $\delta(S)$ et $\mathcal{L}$ soient le plus petites possible, et que $deg(S)$ soit le plus grand possible.

La wide-trail strategy permet de dire plus précisément comment la sécurité d'un chiffrement par blocs dépend de la sécurité de la boîte-S.

1.3.6.1 Propriétés différentielles des boîtes-S

D'après la wide-trail strategy, l'objectif est d'avoir $\delta(S)$ minimale. Nous verrons ici les conditions que cela implique ainsi qu'un bref état de l'art sur la résistance différentielle des boîtes-S.

Définition 1.11 (Boîte-S Parfaitement Non-linéaire (PN))**.** Optimalement, on voudrait $\delta_S(a,b) = \frac{\#A}{\#B}$, $\forall a \in A^*$, $\forall b \in B$. Remarquons que pour certains choix de groupes, ce minimum n'est pas atteignable. Les boîtes-S qui atteignent cette borne sont dites *Parfaitement Non-linéaires* (PN).

Propriété 1.2 (Caractérisation par la dérivée)**.** *Soient deux groupes $(A,\oplus)$ et $(B,\boxplus)$. Si ces groupes sont de même ordre, alors une boîte-S est PN si et seulement si ses dérivées en tout point (non-nul) sont bijectives.*

Démonstration. Il s'agit de remarquer qu'une boîte-S S est PN si $\delta_S(a,b) = \frac{\#A}{\#B}$, $\forall a \in A^*$, $\forall b \in B$, or $\#A = \#B$, donc S est PN si et seulement si $\delta_S(a,b) = 1$, $\forall a \in A^*$, $\forall b \in B$, ce qui signifie simplement que pour tout $a \in A^*$, $\delta_S(a,.)$ est une bijection. □

Propriété 1.3 (Non-optimalité dans $(\{0,1\}^n, \mathsf{XOR})$)**.**
Aucune fonction n'est PN de $(\{0,1\}^n, \mathsf{XOR})$ dans $(\{0,1\}^n, \mathsf{XOR})$.

Démonstration. Il s'agit d'une simple conséquence de la Propriété précédente : puisque pour toute f de $(\{0,1\}^n,\oplus)$ dans $(\{0,1\}^n,\oplus)$, toutes les valeurs de la DDT sont paires, elles ne peuvent pas être égales à $\frac{\#\{0,1\}^n}{\#\{0,1\}^n} = 1$. □

Puisqu'il n'existe pas de fonction Parfaitement Non-linéaire sur le cas usuel $(\{0,1\}^n, \mathsf{XOR})$, on y définit les fonctions optimales (Presque Parfaitement Non-linéaires).

Définition 1.12 (APN)**.** Soit S une fonction de $(\{0,1\}^n, \mathsf{XOR})$ dans $(\{0,1\}^n, \mathsf{XOR})$. Alors $\delta(S) \geqslant 2$ et les fonctions qui atteignent cette borne sont appelées Presque Parfaitement Non-linéaires (APN).

Propriété 1.4 (Caractérisation APN par la dérivée)**.** *Soit S une fonction de $(\{0,1\}^n, \mathsf{XOR})$ dans $(\{0,1\}^n, \mathsf{XOR})$. S est APN si et seulement si quel que soit $a \in \{0,1\}^n$, $a \neq 0$, $\#D_aS(F_2^n) = 2^{n-1}$.*

Démonstration. Cette caractérisation est élémentaire. Une fonction S est APN si et seulement si $\delta_S(a,b) \in \{0,2\}$ pour tout a non-nul et pour tout b. Or $\sum_b \delta_S(a,b) = 2^n$ par définition de $\delta_S(a,b)$, ce qui veut bien dire que pour tout a, $\delta_S(a,.)$ vaut 0 la moitié du temps et 2 l'autre moitié. Par définition, $D_a(S)$ ne peut pas prendre les valeurs où $\delta_S(a,.)$ vaut 0, donc $D_a(S)$ prend exactement 2^{n-1} valeurs. □

Propriété 1.5 (Caractérisation par la dérivée seconde)**.**
Soit S une fonction de $(\{0,1\}^n, \mathsf{XOR})$ dans $(\{0,1\}^n, \mathsf{XOR})$. S est APN si et seulement si quel que soit $a \in \{0,1\}^n$, $a \neq 0$, quel que soit $b \in \{0,1\}^n$, $a \neq b$ et quel que soit $x \in \{0,1\}^n$, $D_b D_a S(x) \neq 0$.

Démonstration. En effet, la dérivée seconde est par définition

$$D_b D_a S(x) = S(x) \oplus S(x \oplus a) \oplus S(x \oplus b) \oplus S(x \oplus a \oplus b)$$

et donc la dérivée seconde s'annule si et seulement si

$$S(x) \oplus S(x \oplus a \oplus b) = S(x \oplus a) \oplus S(x \oplus b)$$

ce qui signifie que la dérivée seconde s'annule si et seulement s'il existe $c \in \{0,1\}^n$ tel que $S(x) \oplus S(x \oplus a \oplus b) = c$ et $S(x \oplus a) \oplus S(x \oplus b) = c$. Ceci équivaut à avoir d'une part x et $x \oplus a \oplus b$ solutions de $D_{a \oplus b} S(X) = c$ et d'autre part $x \oplus a$ et $x \oplus b$ solutions de la même équation différentielle, ce qui veut dire que $\delta_S(a \oplus b, c) \geqslant 4$. □

Corollaire 1.1. *Soient deux groupes $(A, \oplus)$ et $(B, \boxplus)$ et $F : A \to B$. Si $(A, \oplus)$ et $(B, \boxplus)$ sont abéliens, alors quel que soit $a \in A^*$, quel que soit $b \in A^*$, $a \neq b$, s'il existe $x \in A$ tel que $D_b D_a F(x) = 0$, alors il existe $c \in B$ tel que $\delta_F(a \oplus b, c) \geqslant 4$.*

Démonstration. Il s'agit simplement de remarquer que la seule condition nécessaire à la preuve précédente est que la loi de groupe soit commutative. La contraposée de la propriété donne alors ce corollaire. □

État de l'art sur les boîte-S d'un point de vue différentiel. Pour tout le reste de cette thèse, on considérera que les corps d'entrée et de sortie $(A, \oplus)$ et $(B, \boxplus)$ sont égaux à $(\{0,1\}^n, \mathsf{XOR})$. Ainsi, la notation $\oplus$ désignera désormais le XOR bit à bit sur des mots de n bits.

Il convient tout d'abord d'évoquer un problème ouvert dont la résolution simplifierait grandement l'analyse différentielle :

Problème ouvert. Une question majeure est d'être capable, à partir d'une table des différences T, de trouver une fonction F telle que $\mathsf{DDT}(F) = T$, ce qui revient à définir une sorte de primitive de la DDT.

Résumé des résultats connus sur les fonctions APN.

- pour tout n non-nul, il existe des fonctions APN dans $(\{0,1\}^n, \mathsf{XOR})$,
- pour tout n impair, il existe des permutation APN dans $(\{0,1\}^n, \mathsf{XOR})$,
- pour $n = 4$, il n'existe pas de permutation APN,
- pour n pair, il a longtemps été conjecturé qui'il n'existe pas de permutation APN. Pourtant, en 2010, Dillon *et al.* [Bro+10] ont exhibé une permutation APN pour $n = 6$. Les tailles paires (et en particulier les tailles puissances de 2) étant les plus utiles en pratique, il existe un grand problème ouvert :

Grand problème APN.
Existe-t-il des permutation APN de $(\{0,1\}^n, \mathsf{XOR})$ dans $(\{0,1\}^n, \mathsf{XOR})$, n pair, en-dehors de la boîte-S de Dillon (et de ses équivalents affines) ?

La section 2.3 poursuit l'étude des généralisations de la boîte-S de Dillon en vue d'un jour parvenir à résoudre ce problème.

Les boîtes-S sur 8 bits. Pour $n = 8$, la meilleure permutation connue est la boîte-S utilisée dans AES. Elle est construite à partir de la permutation $x \mapsto x^{-1}$ et est d'uniformité différentielle 4. Il s'agit de la seule permutation de 8 bits (à équivalence affine près) d'uniformité différentielle aussi basse qui soit connue.

1.3.6.2 Propriétés linéaires des boîtes-S

Tout comme pour l'uniformité différentielle, certaines bornes générales sont connues pour la linéarité [Rot76 ; Nyb91a ; CV95].

Propriété 1.6 (Fonctions courbes)**.** *Soit S une fonction de $(\mathbb{F}_2^n, \oplus, .)$ dans $(\mathbb{F}_2^m, \oplus, .)$. Alors*

$$\mathcal{L}(S) \geqslant 2^{\frac{n}{2}}.$$

Les fonctions atteignant cette borne sont appelées courbes.

Propriété 1.7 (Fonctions presque courbes)**.** *Soit S une fonction de $(\mathbb{F}_2^n, \oplus, .)$ dans $(\mathbb{F}_2^n, \oplus, .)$. Alors*

$$\mathcal{L}(S) \geqslant 2^{\frac{n+1}{2}}.$$

Les fonctions atteignant cette borne sont appelées presque courbes.

Il n'existe pas de fonctions presque courbes (AB) pour n pair.

1.3.6.3 Liens entre la résistance aux attaques différentielles et linéaires

Les attaques différentielles et linéaires exploitent toutes deux la proximité de la fonction à une fonction linéaire. Ainsi, l'uniformité différentielle et la linéarité sont deux mesures différentes de la distance entre f et les fonctions linéaires.

Sans surprise, ces deux valeurs sont donc liées.

Propriété 1.8 (courbe $\Leftrightarrow$ PN [MS89 ; Nyb91a])**.** *Soit S une fonction de $(\mathbb{F}_2^n, \oplus, .)$ dans $(\mathbb{F}_2^m, \oplus, .)$. Alors S est courbe si et seulement si S est parfaitement non-linéaire. De plus, de telles fonctions n'existent que pour n pair et $n \geqslant 2m$.*

Propriété 1.9 (AB $\Rightarrow$ APN, [CV95])**.** *Soit S une fonction de $(\mathbb{F}_2^n, \oplus, .)$ dans $(\mathbb{F}_2^n, \oplus, .)$, n impair. Si S est presque courbe, alors S est presque parfaitement non-linéaire.*

Propriété 1.10 (DDT et LAT, [Can06 ; Can+01a])**.** *⌋ Soit S une fonction de $(\mathbb{F}_2^n, \oplus, .)$ dans $(\mathbb{F}_2^n, \oplus, .)$. Alors la* DDT *de S détermine la* LAT^2 *de S et réciproquement.*

Démonstration. Il s'agit d'exploiter des propriétés bien connues de la transformée de Walsh, en particulier la relation de Parseval.

Considérons une fonction booléenne vectorielle F de $\mathbb{F}_2^n$ dans $\mathbb{F}_2^n$.

$$
\begin{aligned}
(\hat{f}(a,b))^2 &= (\hat{f}_b(a))^2 \left(\sum_{x\in\mathbb{F}_2^n} (-1)^{b\cdot F(x)+a\cdot x} \right)^2 \\
&= (\hat{f}_b(a))^2 \sum_{x\in\mathbb{F}_2^n} \sum_{y\in\mathbb{F}_2^n} (-1)^{b\cdot[F(x)+F(y)]+a\cdot x+a\cdot y} \\
&= (\hat{f}_b(a))^2 \sum_{x\in\mathbb{F}_2^n} \sum_{d\in\mathbb{F}_2^n} (-1)^{F(x)+F(x+d)+a\cdot x+a\cdot x+a\cdot d} \\
&= (\hat{f}_b(a))^2 \sum_{x\in\mathbb{F}_2^n} \sum_{d\in\mathbb{F}_2^n} (-1)^{F(x)+F(x+d)+a\cdot d} \\
&= (\hat{f}_b(a))^2 \sum_{x\in\mathbb{F}_2^n} \sum_{d\in\mathbb{F}_2^n} (-1)^{D_dF(x)+a\cdot d} \\
&= (\hat{f}_b(a))^2 \sum_{d\in\mathbb{F}_2^n} (-1)^{a\cdot d} \left(\sum_{x\in\mathbb{F}_2^n} \sum_{d\in\mathbb{F}_2^n} (-1)^{D_dF(x)} \right) \\
&= (\hat{f}_b(a))^2 \sum_{d\in\mathbb{F}_2^n} (-1)^{a\cdot d} \mathcal{F}(D_dF)\,,
\end{aligned}
$$

où $\mathcal{F}(D_dF)$ est la transformée de Fourier de D_dF.

Ainsi, par bijectivité de la transformée de Fourier, la DDT de F et la LAT^2 de F sont en bijection. □

Remarque 1.7 (De la validité de δ et $\mathcal{L}$ comme critères de sécurité.). J'aimerais noter ici que, si l'uniformité différentielle δ et la linéarité $\mathcal{L}$ sont des critères de sécurité valides pour les boîtes-S, elles n'en sont pas pour un chiffrement.

En effet, on veut minimiser toutes les valeurs de la DDT et de la LAT d'une boîte-S, ce qui revient à minimiser leurs maxima (δ et $\mathcal{L}$ respectivement).

Su un chiffrement, une clef secrète intervient qui est considérée comme une variable aléatoire. L'approche habituelle consiste alors à étudier le MEDP et le MELP d'un chiffrement $E = (E_k)_{k\in\mathbb{F}_2^K}$ à valeurs dans $\mathbb{F}_2^n$:

$$\mathsf{MEDP}(E) = \max_{a\neq 0,b} \mathsf{Moy}_{k\in\mathbb{F}_2^K} \frac{\delta_{E_k}(a,b)}{2^n}\,,$$

$$\mathsf{MEDP}(E) = \max_{a\neq 0,b} \mathsf{Moy}_{k\in\mathbb{F}_2^K} \frac{\hat{E_k}(a,b)}{2^n}\,.$$

Or, au niveau d'un chiffrement, il n'y a pas de raison particulière de considérer les maxima des distributions. L'objectif pour un chiffrement est de se comporter comme une permutation aléatoire P, or les distributions des DDT et LAT d'une permutation aléatoire ne minimisent pas $\delta_P(a,b)$ et $\hat{P}(a,b)$.

Ainsi, les maxima MEDP et MELP (liés aux maxima δ et $\mathcal{L}$) ne capturent pas entièrement la résistance d'un chiffrement aux attaques différentielles et linéaires. A priori, il est nécessaire d'étudier la distribution entière, on ne peut pas se contenter du maximum, même si, en pratique, il est déjà compliqué d'étudier le maximum.

En revanche, on pourrait avec autant de validité considérer d'autres statistiques que le maximum. À titre d'idée, une autre approximation de la sécurité d'un chiffrement face aux attaques différentielles (resp. linéaires) pourrait être de mesurer la moyenne et la variance d'une différentielle (resp. d'un biais linéaire) non-nulle du chiffrement et de les comparer à l'espérance et la variance attendues pour une permutation aléatoire. Une autre approche équivalente pourrait être de compter la moyenne et la variance du nombre de différentielles nulles.

1.3.6.4 État de l'art sur les boîtes-S

Résumons ici l'état de l'art en termes de constructions de boîtes-S avant ma thèse. Il s'agit d'un sujet largement étudié mais sur lequel beaucoup de questions restent encore sans réponse.

En particulier, l'étude des boîtes-S avec une focalisation sur le bas coût avait jusque là été peu étudié. La plupart des boîtes-S connues offrant de bonnes propriétés cryptographiques étaient jusqu'alors des fonctions puissances, *i.e.* de la forme $x \mapsto x^e$, pour un certain exposant e, de $\mathbb{F}_2^n$ dans $\mathbb{F}_2^n$.

Je résume ici les résultats connus sur les fonctions puissances[24]. D'autres résultats intéressants sur les boîtes-S en dimension petite obtenus par des recherches exhaustives sont résumés en section 2.2.1.

Les boîtes-S en dimension impaire. En dimension impaire ($n = 2t + 1$), on connaît des fonctions puissances APN. Une liste non-exhaustive de ces fonctions est donnée dans la table 1.1. On connaît également des fonctions puissances AB, dont quelques unes sont listées dans la table 1.2.

Table 1.1 – Fonctions puissances APN connues de la forme $x \mapsto x^e$ de $\mathbb{F}_2^n$ dans $\mathbb{F}_2^n$ avec $n = 2t + 1$.

Fonctions	Exposant e	Ref.
fonctions quadratiques $\mathcal{Q}(i)$	$2^i + 1$, avec $\gcd i, n = 1$, $1 \leqslant i \leqslant t$	[Gol68 ; Nyb94a]
fonctions de Kasami $\mathcal{K}(i)$	$2^{2i} - 2^i + 1$, avec $\gcd i, n = 1$, $2 \leqslant i \leqslant t$	[Kas71]
fonction de Welch	$2^t + 3$	[Dob99b ; CCD00]
fonction de Niho	$2^t + 2^{\frac{t}{2}} - 1$ si t est pair, $2^t + 2^{\frac{3t+1}{2}} - 1$ si t est impair	[Dob99a ; HX01]
fonction inverse	$2^{2t} - 1$	[Nyb94a ; BD94]
fonction de Dobbertin	$2^{4g} + 2^{3g} + 2^{2g} + 2^g - 1$, avec $n = 5g$	[Dob01]

Les boîtes-S en dimension paire. En dimension paire $n = 2t$, on ne connaît qu'une seule permutation APN (à équivalence CCZ près, cf. section 2.1.3.2). Les meilleures boîtes-S pour la résistance aux attaques différentielles vérifient donc $\delta = 4$. La littérature est riche en constructions de fonctions avec $\delta = 4$ pour n pair.

De plus, on ne connaît pas de fonction en dimension paire avec $\mathcal{L} \leqslant 2^{\frac{n}{2}+1}$ (il est d'ailleurs conjecturé qu'il n'existe pas de fonction puissance vérifiant $\mathcal{L} \leqslant 2^{\frac{n}{2}+1}$). Rappelons qu'on sait que $\mathcal{L} \geqslant 2^{\frac{n+1}{2}}$. L'existence d'une fonction avec $2^{\frac{n+1}{2}} \leqslant \mathcal{L} \leqslant 2^{\frac{n}{2}+1}$ est une question ouverte. En particulier, on s'intéresse donc de près aux boîtes-S avec $\mathcal{L} = 2^{\frac{n}{2}+1}$.

Je ne citerai ici que les exemples connus qui sont des fonctions puissances : avec $\delta = 4$ dans la table 1.3 et avec $\mathcal{L} = 2^{\frac{n}{2}+1}$ dans la table 1.4. Nous verrons dans la section 2.3 une façon de créer de telles fonctions qui ne sont pas des fonctions puissances.

24. Ces tables sont inspirées de [Can06].

Table 1.2 – Fonctions puissances AB connues de la forme $x \mapsto x^e$ de $\mathbb{F}_2^n$ dans $\mathbb{F}_2^n$ avec $n = 2t + 1$.

Fonctions	Exposant e	Ref.
fonctions quadratiques $\mathcal{Q}(i)$	$2^i + 1$, avec $\gcd i, n = 1$, $1 \leqslant i \leqslant t$	[Gol68 ; Nyb94a]
fonctions de Kasami $\mathcal{K}(i)$	$2^{2i} - 2^i + 1$, avec $\gcd i, n = 1$, $2 \leqslant i \leqslant t$	[Kas71]
fonction de Welch	$2^t + 3$	[Dob99b ; CCD00]
fonction de Niho	$2^t + 2^{\frac{t}{2}} - 1$ si t est pair, $2^t + 2^{\frac{3t+1}{2}} - 1$ si t est impair	[Dob99a ; HX01]

Table 1.3 – Fonctions puissances connues de la forme $x \mapsto x^e$ de $\mathbb{F}_2^n$ dans $\mathbb{F}_2^n$ vérifiant $\delta = 4$ avec $n = 2t$.

n	Exposant e	e^{-1}
$n = 8$	127	127
$n = 10$	5	205
	17	181
	13	79
	103	149
	223	367
	511	511
$n = 12$	73	731
	2047	2047

1.3.7 Attaques algébriques

Le troisième type de distingueur très utilisé en pratique est le distingueur algébrique. Je ne rentrerai pas ici dans beaucoup de détails sur ce type de distingueur car ma thèse se focalisera sur les distingueurs différentiel et linéaire dans un premier temps, mais une brève introduction aux différentes représentations algébriques s'impose. Un exemple d'attaque algébrique est donné dans mes travaux sur la cryptanalyse de FLIP (cf. Chapitre 4)

Les attaques algébriques consistent à exploiter une propriété « non-aléatoire » dans la représentation sous forme de polynôme multivarié du chiffrement. Par exemple, si le chiffrement a un degré algébrique peu élevé, on pourra retrouver de l'information en résolvant un système d'équations non-linéaires. La complexité de l'attaque consiste en le temps de résolution de ce système, qui dépend directement du degré algébrique des équations. Ainsi, plus le degré sera élevé, plus l'attaque sera difficile pour l'adversaire.

Définition 1.13 (Représentation en polynôme multivarié (ANF))**.** Toute fonction booléenne f à n variables peut être représentée par un unique polynôme multivarié à coefficients

Table 1.4 – Fonctions puissances connues de la forme $x \mapsto x^e$ vérifiant $\mathcal{L} = 2^{\frac{n}{2}+1}$ de $\mathbb{F}_2^n$ dans $\mathbb{F}_2^n$ avec $n = 2t$.

Exposant e	Condition sur $n = 2t$	Ref.
$2^{n-1} - 1$		[LW90]
$2^i + 1$, avec $\gcd i, n = 2$	$n = 2 \bmod 4$	[Gol68 ; Nyb94a]
$2^{2i} - 2^i + 1$, avec $\gcd i, n = 2$	$n = 2 \bmod 4$	[Kas71]
$\sum_{i=0}^{n/2} 2^{ik}$, avec $\gcd k, n = 1$	$n = 0 \bmod 4$	[Dob98]
$2^{\frac{n}{2}} + 2^{\frac{n+2}{4}} + 1$	$n = 2 \bmod 4$	[CD96]
$2^{\frac{n}{2}} + 2^{\frac{n}{2}-1} + 1$	$n = 2 \bmod 4$	[CD96]
$2^{\frac{n}{2}} + 2^{\frac{n}{4}} + 1$	$n = 4 \bmod 8$	[Dob98]

dans $\mathbb{F}_2$ qu'on nomme généralement *forme algébrique normale* (ANF) de f, défini par :

$$f(x_0, \ldots, x_{n-1}) = \sum_{u \in \mathbb{F}_2^n} a_u \left(\prod_{i=0}^{n-1} x_i^{u_i} \right) ,$$

où u_i est le i-ème bit de u.

Définition 1.14 (Degré algébrique)**.** Soit f une fonction booléenne à n variables. On appelle *degré algébrique* de f noté $deg(f)$ le degré de sa forme algébrique normale, c'est-à-dire le maximum des poids de Hamming des degrés de l'ANF de f :

$$deg(f) = \max_{u \in \mathbb{F}_2^n} \{ w_H(u) | a_u \neq 0 \} ,$$

où a_u sont les coefficients de l'ANF de f.

Soit F une fonction booléenne vectorielle de $\mathbb{F}_2^n$ dans $\mathbb{F}_2^n$, *i.e.* une fonction de $\mathbb{F}_2^n$ dans $\mathbb{F}_2^n$. On appelle *degré algébrique* de F noté $deg(F)$ le maximum des degrés algébriques de ses fonctions coordonnées $f_i : (x_1, \ldots, x_n) \mapsto (F(x_1, \ldots, x_n))_i$, $1 \leqslant i \leqslant n$, avec $(F(x_1, \ldots, x_n))_i$ le i-ème bit de $F(x_1, \ldots, x_n)$.

Définition 1.15 (Représentation multivariée)**.** Une fonction booléenne vectorielle sur l'espace vectoriel $\mathbb{F}_2^n$ peut être représentée de manière unique par une collection de n polynômes multivariés à coefficients binaires correspondant aux formes normales algébriques de ses coordonnées.

Définition 1.16 (Représentation univariée)**.** Alternativement, on peut identifier l'espace vectoriel $\mathbb{F}_2^n$ au corps fini $\mathbb{F}_{2^n}$. Dans ce cas, une fonction F est vue comme une application univariée de $\mathbb{F}_{2^n}$ dans lui-même $x \mapsto F(x) = y$.

On peut alors représenter F de manière unique par un polynôme univarié à coefficients dans $\mathbb{F}_{2^n}$.

Définition 1.17 (Degré univarié)**.** Le *degré univarié* d'une fonction booléenne vectorielle F de $\mathbb{F}_2^n$ dans $\mathbb{F}_2^n$ est le degré du polynôme univarié dans $\mathbb{F}_{2^n}[X]$ représentant F lorsqu'on identifie F à une fonction de $\mathbb{F}_{2^n}$ dans $\mathbb{F}_{2^n}$.

Exemple 1.1. La fonction $x \mapsto x^3$ dans $\mathbb{F}_{2^n}$ est de degré univarié 3 et de degré algébrique 2 (quadratique). Plus généralement, le polynôme univarié $x \mapsto x^e$ de $\mathbb{F}_{2^n}$ est de degré univarié e et son degré algébrique est le poids de Hamming de e.

Sans rentrer dans les détails, si le chiffrement a un degré algébrique assez élevé, un attaquant collectera des équations de degré élevé et ne sera donc pas en mesure de résoudre son système d'équations en un temps raisonnable. Dans un SPN, on peut directement estimer le degré algébrique du chiffrement à partir du degré algébrique de la boîte-S comme $deg(E) \simeq \min\{(deg(S))^r, 2^n - 1\}$, où r est le nombre de tours. Il faudra donc que le degré algébrique de la boîte-S soit élevé.

1.4 Authentification par les MAC

Si le premier objectif de la cryptographie est la confidentialité, son deuxième est l'authenticité. La solution de la cryptographie symétrique pour assurer l'authenticité d'un message est le MAC (Message Authentication Code).

Il s'agit d'une fonction F qui prend en entrée une clef et un message de taille quelconque et qui produit une sortie de taille fixée appelée « étiquette » ou MAC. On veut qu'il soit difficile de trouver un couple de message m et d'étiquette t tel que, pour une clef aléatoire, $F(k, m) = t$. Dans ce cas, l'étiquette, envoyée avec le message, permet d'assurer l'authenticité du message.

Je ne rentrerai pas ici dans plus de détails sur les MACs, ce sujet est élaboré au chapitre 5, dans lequel nous étudions la sécurité de plusieurs constructions théoriques de MACs avant de développer et d'implémenter un MAC à vas coût pour micro-contrôleurs 32 bits.

1.5 La cryptographie aujourd'hui

Si la cryptographie existe depuis l'aube des temps, elle a vu son essor au XX$^{\text{ème}}$ siècle et se développe tout particulièrement depuis les débuts de la cryptographie civile dans les années 1970.

Si les fondements de la cryptographie symétrique sont des théories mathématiques très poussées fondées sur les probabilités, les statistiques et l'algèbre dans les corps finis, une grande part de la recherche actuelle en cryptographie symétrique s'intéresse aujourd'hui à proposer des solutions adaptées pour des cas d'usages particuliers, tels que la cryptographie pour micro-contrôleurs ou la cryptographie pour le calcul distribué ou délégué par exemple.

En effet, les cryptographes disposent aujourd'hui de schémas de construction (Feistel ou SPN pour les chiffrements par blocs par exemple) bien étudiés. L'étude de ces schémas repose sur des modèles d'analyse (wide-trail strategy par exemple, cf. section 1.3.4) qui permettent de réduire les arguments de sécurité d'un schéma à des conditions sur ses composantes (boîte-S et fonction de diffusion pour un chiffrement par blocs).

Néanmoins, il est à noter que ces modèles d'analyse supposent un certain nombre d'hypothèses sur les composantes Par exemple, dans l'étude de la sécurité des modes d'opérations, on fait l'hypothèse que le chiffrement sous-jacent est une permutation aléatoire. Dans l'étude des chiffrements itérés, on fait l'hypothèse que les tours sont indépendants les uns des autres.

Quoi qu'il en soit, suivre ces schémas préexistants permet de construire sans trop d'efforts des primitives symétriques avec des arguments de sécurité solides dans le modèle d'analyse (et donc sous les hypothèses du modèle).

Ainsi, du côté de la création de nouvelles primitives, les fondements mathématiques, déjà bien étudiés, permettent aujourd'hui d'imaginer de tout nouveaux chiffrements sans

avoir toujours à rentrer en détail dans les preuves mathématiques, le gros du travail étant déjà assuré par des travaux précédents. Un grand nombre de nouvelles primitives cherchent ainsi plutôt à optimiser les chiffrements ou à apporter de nouvelles alternatives au standard AES. Ces recherches ne changent pas de modèle d'analyse, mais tentent de pousser le modèle jusqu'à ses limites.

1.5.1 Moins cher sans perte de sécurité, le défi de la cryptographie à bas coût

Le coût de la cryptographie est aujourd'hui largement abordable dans la plupart des situations, même si la cryptographie reste chère : très lente comparée aux autres parties d'un protocole Internet, demandant beaucoup d'espace sur les puces pour des implémentations matérielles ou consommant beaucoup d'énergie dans des objets fonctionnant sur pile, par exemple.

Minimiser les coûts (en temps, en énergie, en espace mémoire, en espace matériel, etc) est donc une problématique importante lorsqu'on a atteint un stade de maturité permettant déjà d'avoir des primitives sûres à des coûts raisonnables : il s'agit maintenant de faire du bas coût. Mais attention, on veut tout de même maintenir la sécurité : là où les primitives plus courantes ont de grandes marges de sécurité, la cryptographie à bas coût va réduire les marges de sécurité pour réduire les coûts, mais sans passer sous un seuil où la primitive deviendrait faible.

D'autre part, la majeure partie des réductions de coûts ne provient pas d'une baisse de sécurité, mais bel et bien de nouvelles idées permettant d'obtenir une sécurité équivalente à un coût moindre. Il peut aussi s'agir de réduire un paramètre de coût (l'espace matériel par exemple), au détriment d'un autre coût (le temps par exemple). Il s'agit alors de bien cibler le cas d'utilisation : lorsqu'on optimise, on ne crée pas des primitives génériques mais des primitives spécifiques à une utilisation ciblée.

1.5.2 Résistance aux attaques par canaux cachés

L'importance de la résistance aux attaques par distingueurs a déjà été détaillée en section 1.3. En revanche, faisons ici un point sur les attaques par canaux cachés. Généralement, on a tendance à ne considérer un chiffrement que du point de vue de son modèle mathématique, c'est-à-dire comme une fonction, à laquelle on peut donner une entrée et dont on peut observer la sortie. On appelle ceci un modèle en « boîte noire », car on ne peut pas observer ce qui se passe à l'intérieur de la fonction : on n'observe que l'entrée et la sortie.

Malheureusement, lorsqu'on passe à la pratique, ce modèle en boîte noire n'est plus suffisant : on peut parfois accéder à l'implémentation et à la machine qui effectue le calcul de la fonction. Ceci donne accès à des observations supplémentaires : temps de calcul ([LN93 ; Koc96]), consommation de courant ([KJJ99]), sons ([AA04]) et bien d'autres encore. Et encore, il s'agit ici seulement d'attaques dites « passives », car l'attaquant ne fait qu'observer le chiffrement de l'extérieur.

S'il est « actif », il peut aussi accéder directement à l'implémentation matérielle du chiffrement et introduire des erreurs dans les calculs pour déduire de l'information ([BS97]). Ainsi, en exploitant l'implémentation, l'attaquant a accès à bien plus d'informations qu'il ne devrait.

La contre-mesure adaptée est de ne pas s'intéresser uniquement à la sécurité apportée par la fonction face aux attaques mathématiques, mais aussi de s'intéresser à la sécurité apportée par l'implémentation face aux attaques physiques.

Il existe aujourd'hui des techniques regroupées sous l'appellation « masquage » qui permettent de cacher ce que fait une fonction en cours de calcul face à certaines attaques

physiques passives ([Cha+99 ; ISW03]). Cependant, le coût d'implémentation de ce masquage est conséquent, et dépend directement du nombre de portes logiques non-linéaires (portes and, or et leurs variantes) présentes dans la représentation logique sous forme de circuit de la fonction à masquer. Il est aussi possible de masquer directement mes opérations dans $\mathbb{F}_{2^n}$. Dans ce cadre, la boîte-S est un composant coûteux à implémenter, puisque c'est le composant non-linéaire du SPN.

Par conséquent, il est essentiel pour résister à certaines attaques par canaux cachés de réduire en premier lieu le nombre d'opérations non-linéaires. Ceci est un problème très différent de ceux des autres requêtes, et il s'agit d'une contrainte supplémentaire. On remarquera en passant que le problème de trouver le circuit le moins coûteux pour une fonction donnée est difficile, au sens NP-complet en théorie de la complexité ([BMP13] pour le cas particulier des fonctions linéaires).

1.5.3 La contribution de ma thèse

Ma thèse s'inscrit dans la recherche en cryptographie symétrique. La thématique centrale de ces recherches est la cryptographie à bas coût. Durant cette thèse, j'ai étudié :

- la construction de composantes de chiffrements par blocs :
  - boîtes-S robustes à bas coût avec des structures de Feistel et de type MISTY (cf. chapitre 2),
  - boites-S presque-optimales avec des structures de Papillon (cf. Chapitre 2),
  - fonctions de diffusion optimales (et presque optimales) à bas coût (cf. chapitre 3),
- la cryptanalyse d'un chiffrement à flot, FLIP, conçu pour des contraintes de bas coût spécifiques au chiffrement complètement homomorphe (cf. chapitre 4),
- la construction d'un MAC à bas coût pour micro-contrôleurs 32 bits, XPMAC, basé sur une fonction de hachage universelle simple et efficace, XPoly (cf. Chapitre 5)
- une réflexion de fond sur quelques directions générales adoptées ces dernières décennies en cryptographie symétrique et de potentielles pistes pour des recherches futures (cf. chapitre 6).

Les travaux sur la construction de boîtes-S robustes à bas coût en exploitant la structure de réseaux de Feistel et de réseaux de type MISTY ont été effectués en collaboration avec Anne Canteaut et Gaëtan Leurent et ont donné lieu à une publication à SAC en 2015 [CDL16]. Ces travaux nous ont permis de construire des boîtes-S de 8 bits robustes et peu coûteuses, adaptées pour une implémentation avec faible surcoût dans le cadre de la résistance aux attaques par canaux auxiliaires. À ce jour, ces boîtes-S sont les plus efficaces sur 8 bits (à sécurité équivalente).

Les travaux sur la structure de Papillon ont un intérêt moins pratique mais important en théorie. Ces travaux ont été effectués avec Anne Canteaut et Léo Perrin et publiés à IEEE-IT en 2017 [CDP17]. Il s'agit de généraliser une structure puissante, le Papillon, qui offre des pistes pour trouver des boîtes-S optimales (pour la résistance aux attaques différentielles) sur un nombre de bits pair, ce qui est un problème fondamental en cryptographie symétrique.

Les travaux sur la construction de fonctions de diffusion optimales à bas coût sont une avancée importante, tant en pratique qu'en théorie. Ces travaux menés avec Gaëtan Leurent et publiés à TosC en 2018 [DL18b], ont permis de développer un algorithme efficace pour trouver des matrices de diffusion sur des tailles pratiques (tailles 3×3 et 4×4 pour des mots de taille quelconque) à un coût très réduit. Dans l'espace de recherche considéré, il s'agit des matrices de diffusion les moins coûteuses.

Précisons que des travaux effectués en parallèle avec une approche différente par Kranz, Leander, Stoffelen et Wiemer [Kra+17] obtiennent des résultats presque aussi bons que les nôtres.

Nos résultats sont significativement meilleurs que toutes les fonctions de diffusion de la littérature d'avant 2017 et offrent la possibilité de construire des chiffrements nettement moins coûteux pour un même niveau de sécurité à l'avenir.

Les travaux de cryptanalyse du chiffrement à flot FLIP ont été effectués conjointement avec Virginie Lallemand et Yann Rotella et publiés à Crypto en 2016 [DLR16]. Dans cet article, nous avons analysé la version soumise à Eurocrypt 2016 [Méa+16] de FLIP, un chiffrement à flot efficace pour le chiffrement complètement homomorphe hybride conçu par Pierrick Méaux, Anthony Journault, François-Xavier Standaert et Claude Carlet. Nous avons formalisé une attaque pratique sur ce chiffrement et l'avons testée en pratique, ce qui a conduit à une mise-à-jour de FLIP (notons que la version finalement publiée à Eurocrypt 2016 résiste à notre attaque).

La conception du MAC pour micro-contrôleurs 32 bits XPMAC a été effectuée avec Gaëtan Leurent et est actuellement soumise à CHES 2018. Sur le plan théorique, ce MAC réutilise des constructions déjà connues, mais il démontre par la pratique qu'il est faisable d'obtenir des MACs extrêmement efficaces basés sur des fonctions de hachage universelles.

D'un point de vue pratique, en revanche, XPMAC constitue un grand pas en avant pour la conception de MACs. En effet, XPMAC s'avère aussi efficace en pratique que Chaskey [Mou+14], le MAC jusqu'ici le moins coûteux, mais à la différence de Chaskey, XPMAC a une preuve de sécurité robuste reposant sur les fonctions de hachage universelles. Un avantage subsidiaire de XPMAC est une extrême simplicité.

Atteindre une telle efficacité, en particulier sur les micro-contrôleurs 32 bits (largement déployés en pratique dans les objets connectés), a requis de nombreuses optimisations, tant au niveau algorithmique qu'au niveau du code. Le résultat est un MAC aux propriétés meilleures que tous les MACs de la littérature.

Ce travail inclut par ailleurs des travaux plus théoriques sur un sujet lié : l'étude des constructions de fonctions de hachage universelles utilisant des composants bijectifs.

Ceci conclut l'introduction générale de cette thèse. Sans plus attendre, entrons désormais dans le vif du sujet avec en premier lieu les études de boîtes-S structurées.

Chapitre 2

Confusion : construction de boîtes-S à bas coût

Blocs et Boîtes : la réunion Tupperware

L'étude de boîtes-S fut la pierre angulaire de ma thèse. Si je me suis intéressé à de nombreux aspects de la cryptographie symétrique, je pense néanmoins que maîtriser la construction de boîtes-S est édifiant. En effet, la boîte-S peut être vue comme une sorte de chiffrement ou de permutation cryptographique miniature : les propriétés désirables des boîtes-S sont presque les mêmes que celles attendues sur les chiffrements et les permutations de grande taille.

Pour cette raison, il me semble que bien comprendre comment construire de bonnes boîtes-S donne aussi une bonne idée de comment construire la majorité des primitives de cryptographie symétrique. La construction de boîte-S peut en ce sens être interprétée comme une étude de fonctions très résistantes sur de petites tailles, et les stratégies de construction de boîtes-S peuvent permettre d'identifier des stratégies de construction de primitives plus grandes.

Dans ce chapitre, je détaille les objectifs dans la recherche de boîtes-S, puis j'expose deux résultats obtenus sur la construction de boîtes-S à bas coût : le premier sur l'étude des réseaux de type Feistel et MISTY [CDL16], le deuxième sur l'étude d'une structure récente appelée Papillon [CDP17].

2.1 Solidifier les fondations : les boîtes-S

Comme précisé en introduction (cf. section 1.3), la confusion apportée par un chiffrement dérive directement de la confusion apportée par ses boîtes-S, avec néanmoins une influence de la partie de diffusion. En pratique, cette confusion est étudiée sous la forme des propriétés distingantes, en particulier l'uniformité différentielle, la linéarité et le degré algébrique. Le modèle d'analyse rigoureux qui réduit la sécurité d'un chiffrement par blocs à des critères sur sa boîte-S et sa matrice de diffusion est appelé *wide-trail strategy* qui établit les critères suivants.

2.1.1 Propriétés désirables

Sans faire de longs discours, la liste des propriétés désirables pour une boîte-S est la suivante :

— Permutation [1] (en particulier, $m = n$)
— Petite taille (n petit)
— Résistance à tous les types d'attaques connus, principalement :
  — Attaques algébriques (degré algébrique élevé)
  — Attaques différentielles (uniformité différentielle basse)
  — Attaques linéaires (linéarité basse)
  — Attaques par invariants (les ensembles algébriques structurés ne sont pas stables par application de la boîte-S)
  — Attaques par canaux cachés (peu d'opérations non-linéaires dans l'implémentation)
— Bas coût

Détaillons chacune de ces requêtes.

La bijectivité n'est pas à proprement parler nécessaire, mais elle est souhaitable car dans certaines applications (en particulier les SPN), la bijectivité de la boîte-S est nécessaire pour avoir la bijectivité du chiffrement (et donc pour pouvoir déchiffrer). Notons que c'est une contrainte forte : la proportion de permutations parmi les fonctions de n bits est petite.

La petite taille n'est pas non plus exactement nécessaire, mais cette contrainte est motivée par le fait qu'on n'a pas de moyen simple d'implémenter ni d'analyser une grande fonction non-linéaire. Ainsi, les tailles de boîtes-S généralement considérées sont 4 et 8 bits. Toutefois, rien n'interdit de considérer des boîtes-S de grande taille hormis la complexité pratique de l'étude et qu'il est difficile de concilier grande taille et faible coût d'implémentation.

2.1.1.1 Représentation des boîtes-S comme circuit binaire

Pour en revenir au sujet principal de ma thèse, qui est la construction à bas coût, les coûts à considérer sont multiples, mais la représentation d'une boîte-S comme un circuit binaire utilisant des portes logiques élémentaires est généralement adaptée, car un petit circuit binaire implique un délai d'exécution court, un coût en mémoire faible, un coût en espace matériel faible et un coût en énergie faible. De plus, comme le détaille la discussion sur les attaques par canaux cachés, cette représentation en circuit binaire est adaptée pour réduire le coût du masquage.

En revanche, là où les portes logiques les plus chères pour le masquage (cf. section 1.5.2) sont les portes non-linéaires (and et or), sans masquage, les portes les plus chères sont les portes linéaires (XOR). En partant du principe que la boîte-S devra sans doute être masquée, il faut donc en priorité baisser le nombre de portes and et or, puis baisser le nombre de portes XOR, et enfin, dans des implémentations logicielles en particulier, réduire le nombre d'opérations de déplacement mov.

Si cette représentation en circuit binaire est adaptée pour l'étude de coût, ce n'est généralement pas celle qui a été choisie par le passé pour construire des boîtes-S résistantes aux attaques statistiques. Il est plus habituel de représenter une fonction linéaire par une matrice et une fonction non-linéaire par son ANF ou sa table des valeurs. Le lien avec la représentation en circuit binaire n'est pas direct, et encore une fois, trouver un circuit optimal pour une fonction donnée est difficile.

2.1.2 Les limites des boîtes-S existantes

J'introduis ici un résumé des travaux existants sur la construction de boîtes-S.

1. Fonction bijective

2.1.2.1 Problèmes d'optimisation complexes

Dimension. Notons tout d'abord qu'on n'arrive à pousser les recherches exhaustives de boîtes-S que jusqu'à des dimensions petites (éventuellement en dimension 6). Or on utilise fréquemment (pour des raisons d'implémentation) des boîtes-S en dimensions 4 ou 8. Il n'existe pas de permutation APN en dimension 4, ce qui rend la dimension 4 sous-optimale. D'un autre côté, la dimension 8 est hors d'atteinte des recherches exhaustives [2].

Coût. L'inconvénient de commencer par chercher des boîtes-S avec de bonnes propriétés cryptographiques et d'optimiser leur coût d'implémentation en second lieu, c'est qu'il est difficile d'optimiser le coût d'une fonction donnée. On en arrive souvent à des implémentations de boîtes-S coûteuses.

2.1.2.2 Problématique : la qualité à bas coût

Comme l'établit le paragraphe précédent, les propriétés désirables pour les boîtes-S consistent à minimiser des propriétés distingantes et des propriétés de coût. La complexité majeure pour exhiber de « bonnes » boîtes-S (qui optimisent tous les paramètres) vient de la réunion de toutes ces conditions.

En effet, si obtenir une boîte-S avec une bonne résistance à toutes les propriétés distingantes à la fois n'est pas impossible a priori, cela reste difficile à mettre en œuvre. Pourtant les propriétés distingantes sont les plus simples à concilier : on cherche un même objet pour toutes ces propriétés, en quelque sorte la fonction la plus non-linéaire possible (les propriétés offrent chacune une définition différente – et non-équivalente – de la non-linéarité).

En revanche, concilier la résistance aux propriétés distingantes et le coût minimal est impossible [3], tout au plus on peut tenter des compromis entre une résistance correcte et un coût abordable.

Forts de ces observations, nous pouvons tenter d'aller plus loin. Ce fut mon sujet d'étude le plus récurrent durant ma thèse. Les sections 2.2, 2.3 et 2.4.1 décrivent les résultats principaux que j'ai obtenus sur le sujet et correspondent à deux articles ([CDL16 ; CDP17]), publiés respectivement à SAC en 2015 et à IEEE-IT en 2016.

L'idée principale de ces constructions de boîtes-S est d'introduire de la structure pour réduire les coûts d'implémentation. En particulier, nous utiliserons les réseaux de Feistel et de type MISTY, jusqu'alors utilisés pour la construction de chiffrements par blocs.

2.1.2.3 Un modèle d'analyse des structures inadapté aux boîtes-S

L'objectif de cette section est de décrire les techniques d'analyse de sécurité des chiffrements par blocs, de compiler les résultats connus pour la sécurité des réseaux de Feistel et de type MISTY pour la construction de chiffrements par blocs, puis de montrer pourquoi ces techniques ne sont pas adaptées à l'analyse des boîtes-S (et même peu adaptées à l'analyse des chiffrements), pour enfin exhiber un modèle fiable pour l'analyse des boîtes-S.

MEDP/MELP/EMDP/EMLP. Rappelons qu'un chiffrement par blocs est une famille de permutations paramétrée par une clef secrète et, plus important ici, *aléatoire*.

Rappelons les critères de mesure de la résistance d'un chiffrement par blocs aux attaques différentielles et linéaires : MEDP et MELP.

2. Il y a 2^{2048} fonctions en dimension 8 et à peu près 2^{1689} permutations, par rapport à 2^{64} fonctions et 2^{44} permutations en dimension 4.

3. Ne serais-ce que parce que le coût minimal est celui de la fonction identité, qui bien évidemment est la pire en terme de sécurité.

Définition 2.1 (MEDP)**.** Soit $(E_k)_{k\in\mathbb{F}_2^K}$ un chiffrement par blocs. On appelle MEDP (Maximum Expected Differential Probability, *i.e.* probabilité différentielle moyenne maximale) de $(E_k)_{k\in\mathbb{F}_2^K}$ la valeur :

$$\mathsf{MEDP}((E_k)_{k\in\mathbb{F}_2^K}) = \max_{a\neq 0,b} \mathsf{Moy}_{k\in\mathbb{F}_2^K} \frac{\delta_{E_k}(a,b)}{2^n} .$$

De même, on définit MELP :

Définition 2.2 (MELP)**.** Soit $(E_k)_{k\in\mathbb{F}_2^K}$ un chiffrement par blocs. On appelle MELP (Maximum Expected Linear Potential), *i.e.* potentiel linéaire moyen maximal) de $(E_k)_{k\in\mathbb{F}_2^K}$ la valeur :

$$\mathsf{MELP}((E_k)_{k\in\mathbb{F}_2^K}) = \max_{a\neq 0,b} \mathsf{Moy}_{k\in\mathbb{F}_2^K} \left(\frac{\hat{E_k}(a,b)}{2^n}\right)^2 .$$

Le théorème suivant donne les résultats connus pour borner MEDP et MELP dans le cas des réseaux de Feistel et de type MISTY.

Théorème 2.1 (Feistel ou MISTY, en moyenne sur les clefs, [NK95 ; AO97 ; Mat08])**.** *Soient S_1, S_2 et S_3 trois permutations de n bits, $p = \max_i \delta(S_i)/2^n$ et $q = \max_i(\mathcal{L}(S_i)/2^n)^2$. Alors la famille de fonctions $(F_k)_{k=(k_1,k_2,k_3)\in\mathbb{F}_2^{3K}}$ définie par 3 tours de réseau de Feistel ou de type MISTY avec S_i comme fonctions internes vérifie*

$$\mathsf{MEDP}(F_K) \leqslant p^2 \text{ et } \mathsf{MELP}(F_K) \leqslant q^2 .$$

Montrons maintenant que MEDP et MELPne mesurent pas bien la résistance d'un chiffrement aux attaques différentielles et linéaires.

Exemple 2.1. Considérons une famille de permutations $(F_k)_k$ sur 8 bits dépendante d'une clef secrète k, suivant une structure de type MISTY (cf. figure 1.7). Avec pour fonction de tour $R = [10, 7, 9, 6, 0, 1, 5, 11, 3, 14, 8, 2, 12, 13, 4, 15]$, le théorème 2.1 nous garantit que $\mathsf{MEDP}((F_k)_k) \leqslant \frac{16}{256}$, car $\delta(R) = 4$. Or, pour *toute* permutation de cette famille (*i.e.* pour tout choix de k), il existe une différentielle de probabilité $\frac{32}{256}$. Ceci n'est pas contradictoire, car la différentielle qui atteint ce maximum dépend de la clef.

En effet, dans le calcul de MEDP (resp. MELP), la différentielle (resp. l'approximation linéaire) (a, b) est fixée *avant* de considérer tous les choix de clef. Ainsi, MEDP (resp. MELP) calcule la moyenne sur les clefs de la probabilité de chaque différentielle, puis prend le maximum de ces moyennes.

Remarque 2.1 (EMDP et EMLP)*.* Il serait plus judicieux de considérer l'espérance des maxima des probabilités différentielles, ce qui correspondrait à EMDP, en échangeant la moyenne et le maximum. Malheureusement, une telle quantité est plus difficile à calculer : là où MEDP calcule le maximum des probabilités différentielles d'une seule fonction (la fonction moyenne par rapport à la clef), EMDP demande de calculer le maximum des probabilités différentielles pour toutes les fonctions de la famille (*i.e.* pour toute clef), puis d'en faire la moyenne. Calculatoirement, EMDP est donc bien plus complexe à évaluer sans méthode adaptée, mais capture bien mieux la résistance d'un chiffrement aux attaques différentielles (les mêmes considérations sont vraies pour remplacer MELP par EMLP) [4].

Dans le cas de la construction de boites-S, on notera que le problème est encore différent puisqu'il n'y a pas de clef. Considérer des quantités en moyenne sur les clefs n'a donc pas de sens. Les critères de sécurité qui caractérisent effectivement la résistance des boîtes-S aux attaques différentielles et linéaires sont l'uniformité différentielle δ (ou de manière équivalente la probabilité différentielle maximale) et la linéarité $\mathcal{L}$.

4. Je signale ici une confusion entre MEDP et EMDP dans [Mat97] : la définition utilisée correspond à EMDP et EMLP, alors que le résultat des théorèmes s'appliquent à MEDP et MELP

Analyser les boîtes-S : un modèle sans clef. Il faut donc étudier un modèle d'analyse *sans clef* pour les boîtes-S. On remarquera en passant que, puisque la clef est fixée, il s'agit simplement d'une constante, qui ne modifie en rien l'analyse différentielle et l'analyse linéaire. Ainsi, pour le cas des cryptanalyses différentielles et linéaires, le modèle sans clef est équivalent à un modèle *à clef fixée*, ce qui correspondrait à EMDP et EMLP.

En particulier, dans les travaux suivants, nous nous intéresserons aux réseaux de Feistel et de type MISTY pour la construction de boîtes-S. Puisque les résultats de la littérature sur les réseaux de type Feistel et MISTY n'étudiaient que les valeurs de MEDP et MELP, ces études n'analysent pas le modèle sans clef nécessaire pour les boîtes-S et il nous faut donc refaire toute l'analyse des réseaux de Feistel et de type MISTY dans le modèle sans clef.

Notons toute de même l'existence d'un article avant notre article de SAC en 2015 ([CDL16]) qui amorçait déjà l'analyse des réseaux de Feistel sans clef, publié par Li et Wang [LW14].

2.1.3 Techniques d'étude : structures et équivalences

2.1.3.1 Les structures usuelles

Afin de réduire les coûts d'implémentation, on s'intéressera à la construction de boîtes-S avec de bonnes propriétés de résistance aux attaques classiques avec de la structure. En particulier, les structures qui nous intéressent le plus sont celles qui permettent de construire une fonction de taille n à partir de plusieurs fonctions de taille moindre. Les structures que nous utiliserons principalement sont schématisées dans les figures 2.1a, 2.1b et 2.1c.

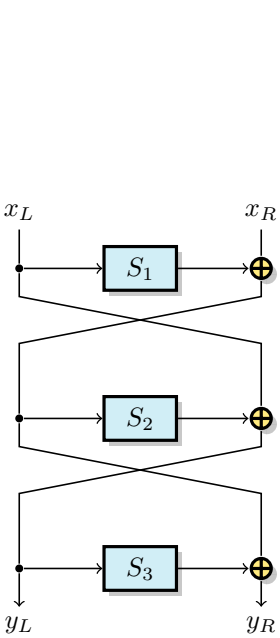


Figure 2.1(a): Schéma de 3 tours d'un réseau de Feistel.

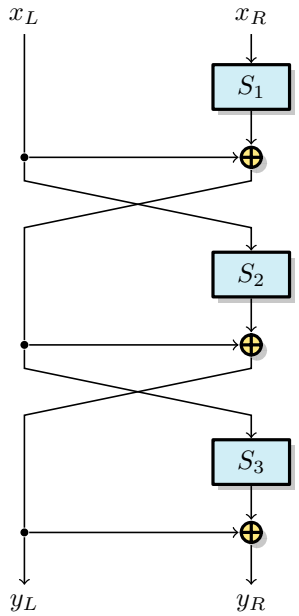


Figure 2.1(b): Schéma de 3 tours d'un réseau de type MISTY.

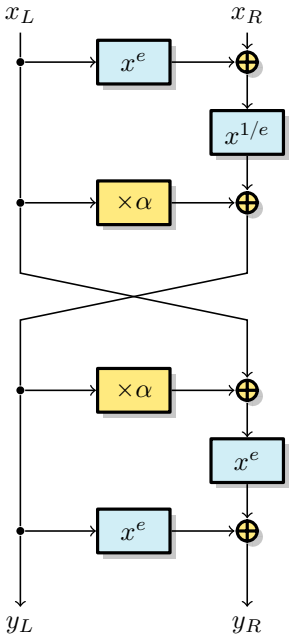


Figure 2.1(c): Schéma d'un Papillon ouvert.

Réseau de Feistel. Le schéma de Feistel (ou réseau de Feistel) a déjà été évoqué pour la construction de chiffrements. C'est en effet un bon candidat pour construire des permuta-

tions non-linéaires.

Une analyse rapide du schéma de Feistel nous donne les résultats suivants. Un schéma de Feistel sépare l'entrée de la fonction en deux moitiés, puis ne fait que des opérations sur les moitiés. Trois tours d'un schéma de Feistel construisent donc une fonction de $2n$ bits à partir de 3 fonctions de n bits (S_1, S_2 et S_3). La fonction construite par un schéma de Feistel est toujours bijective, et même quasi-involutive [5]. En effet, il suffit d'inverser l'ordre de S_1, S_2 et S_3 pour obtenir l'inverse (ce qui revient à lire le schéma de bas en haut). Ceci est très pratique, car on vérifiera toujours la condition de construire une permutation, même si S_1, S_2 et S_3 ne sont pas elles-mêmes bijectives. La propriété de quasi-involutivité permet d'implémenter l'inverse de la boîte-S à faible surcoût (pour le déchiffrement par exemple).

Ainsi, avec L_i la moitié de gauche de l'état au tour i et R_i la moitié de droite, on a :

$$L_{i+1} = R_i \oplus S_i(L_i)\,,$$

$$R_{i+1} = L_i\,.$$

Si l'on considère un tour de schéma de Feistel avec du recul [6], l'opération non-linéaire est la fonction sur n bits S_i. Après un tour de schéma de Feistel, la moitié gauche est laissée inchangée. Quant à la moitié droite, elle dépend de tous les bits d'entrée (diffusion), et dépend non-linéairement des bits de la moitié gauche (la confusion ne dépend que de la moitié des bits).

Réseau de type MISTY. Le schéma de type MISTY (du nom du chiffrement MISTY1 [Mat97]) est similaire au schéma de Feistel. La différence majeure est qu'après un tour de schéma de type MISTY [7], la moitié droite dépend non-linéairement de la moitié droite de l'entrée, et non de la moitié gauche.

Pour un schéma de type MISTY, avec L_i la moitié de gauche de l'état au tour i et R_i la moitié de droite, on a :

$$L_{i+1} = S(R_i) \oplus L_i\,,$$

$$R_{i+1} = L_i\,.$$

Une autre différence notoire est qu'un schéma de type MISTY ne construit une permutation que si S_1, S_2 et S_3 sont des permutations (l'inverse correspond au schéma lu de bas en haut, mais en prenant S_i^{-1} à la place de S_i, pour tout i).

L'intérêt du réseau de type MISTY est qu'il permet d'évaluer S_1 et S_2 en parallèle pour gagner du temps de calcul.

Réseau de type Lai-Massey. Une autre variante du réseau de Feistel existe, dû à Lai et Massey [LM91], qui offre des propriétés similaires. Trois versions de réseaux de type Lai-Massey existent : l'originale de Lai et Massey (qui est équivalente à un réseau de Feistel légèrement généralisé, cf. figure 2.3b), une variante proposée dans le chiffrement Whirlpool [BR+00] et une version allégée de celle de Whirlpool proposée par Pierre Karpman pour construire la boîte-S Littlun [Kar16] (il s'agit d'un travail qui applique les idées de notre article sur les réseaux de Feistel et de type MISTY à une autre structure). Ces trois variantes sont schématisées en figures 2.2a , 2.3a et 2.2c.

La version originale utilise un orthomorphisme σ, *i.e.* une permutation σ dans un groupe tel que $x \mapsto x^{-1}\sigma(x)$ soit une permutation.

5. Une involution est une fonction F telle que $F = F^{-1}$.
6. Sans échange des deux moitiés pour simplifier.
7. Sans échange des deux moitiés.

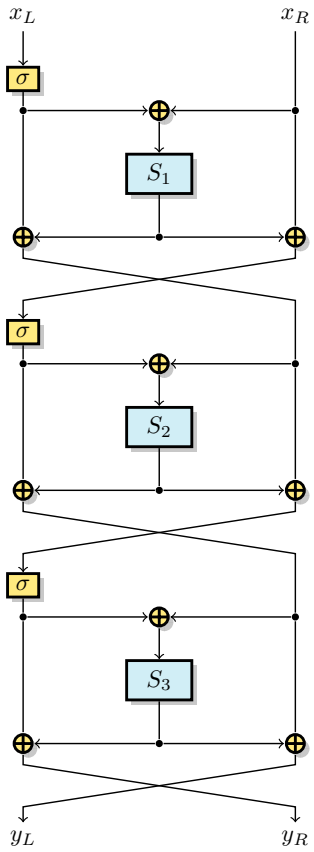


Figure 2.2(a): Schéma de 3 tours de réseau de type Lai-Massey.

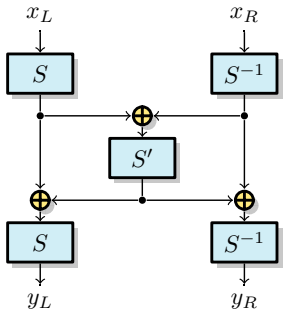


Figure 2.2(b): Schéma de la boîte-S de Whirlpool.

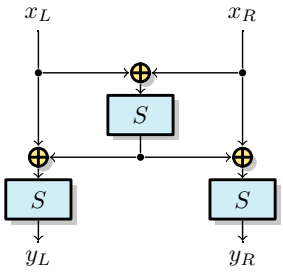


Figure 2.2(c): Schéma de la boîte-S Littlun.

Papillon. Le schéma appelé Papillon est une idée récente de chercheurs de l'Université de Luxembourg, introduite dans [PUB16] alors qu'ils cherchaient la structure de la boîte-S de Dillon [Bro+10], qui est la seule réponse connue (à équivalence près) au Grand Problème APN. Ils ont trouvé que cette permutation de 6 bits APN pouvait être représentée par un Papillon, ce qui rend cette structure particulièrement intéressante puisqu'elle permet de créer des boîtes-S optimalement résistantes aux attaques classiques.

Structurellement, un Papillon correspond à un mélange de schéma de Feistel et de schéma de type MISTY : dans la première moitié (appelons-la R), on enchaîne un tour de schéma de Feistel, puis un tour de schéma de type MISTY avec un ajout d'une multiplication (linéaire) par un scalaire, ensuite, on échange les deux moitiés et on applique l'inverse R^{-1}.

Par ailleurs, pour $\alpha = 1$, le Papillon correspond exactement à 3 tours de Feistel avec les boîtes-S x^e, $x^{\frac{1}{e}}$ et x^e, et une variante du Papillon (étudiée en section 2.3) est alors une construction similaire à la boîte-S Littlun (cf. figure 2.2c).

Après une moitié de Papillon R, la moitié de gauche est laissée inchangée, et la moitié de droite dépend non-linéairement des deux moitiés de l'entrée (elle est donc parfaitement mélangée d'après les critères de Shannon).

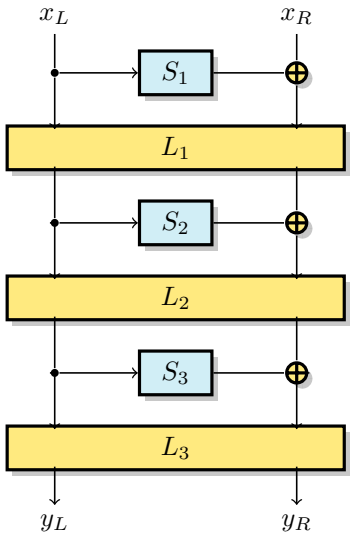


Figure 2.3(a): Schéma de 3 tours d'un réseau de Feistel plus général (la partie linéaire est plus qu'un échange des deux moitiés).

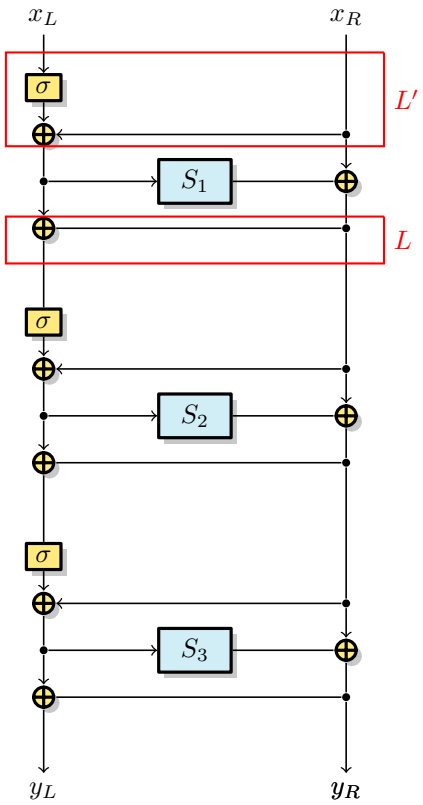


Figure 2.3(b): Représentation de 3 tours d'un réseau de type Lai-Massey comme 3 tours d'un réseau de Feistel plus général. $L_1 = L_2 = L_3 = L' \circ L$.

2.1.3.2 Quelques relations d'équivalence utiles

En parallèle de ces structures, nous utiliserons des relations d'équivalence qui permettront d'étudier des boîtes-S complexes en se ramenant à des équivalents plus simples. Ces équivalences sont les suivantes.

Un type d'équivalence très général est l'équivalence affine [Bir+03]. Pour toute fonction F, un équivalent affine est une fonction de la forme $A_1 \circ F \circ A_2$, où A_1 et A_2 sont des permutations affines, c'est-à-dire $A_1(x) = L_1(x) \oplus c_1$, pour L_1 une fonction linéaire et c_1 une constante (et de même pour A_2). La classe d'équivalence affine d'une fonction F de $\mathbb{F}_2^n$ dans $\mathbb{F}_2^n$ sera donc par définition

$$\{(x \mapsto L_1(F(L_2(x) \oplus c_2)) \oplus c_1 \mid L_1,\, L_2 \in \mathrm{GL}(\mathbb{F}_2^n),\, c_1,\, c_2 \in \mathbb{F}_2^n\}$$

où $\mathrm{GL}(\mathbb{F}_2^n)$ est l'ensemble des permutations linéaires de $\mathbb{F}_2^n$ dans $\mathbb{F}_2^n$. Dans cette thèse, on notera $F \overset{\mathsf{Aff}}{\sim} G$ pour l'équivalence affine entre F et G.

L'intérêt majeur de l'équivalence affine est qu'elle préserve les propriétés cryptographiques essentielles, à savoir l'uniformité différentielle, la linéarité, le degré algébrique et la bijectivité. Formellement,

Propriété 2.1 (Équivalence affine)**.** *Si* $F \overset{\mathsf{Aff}}{\sim} G$*, alors :*
- $\deg(F) = \deg(G)$,
- $\{\{\mathrm{DDT}(F)\}\} = \{\{\mathrm{DDT}(G)\}\}$ *(en particulier,* $\delta(F) = \delta(G)$*),*
- $\{\{\mathrm{LAT}(F)\}\} = \{\{\mathrm{LAT}(G)\}\}$ *(en particulier,* $\mathcal{L}(F) = \mathcal{L}(G)$*).*

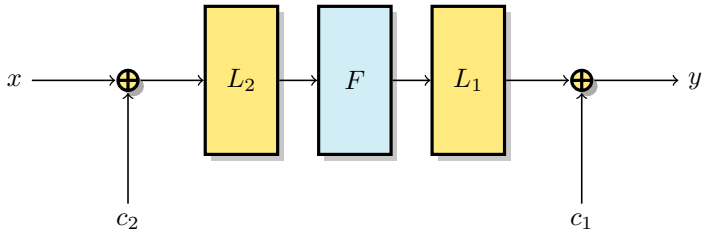


Figure 2.4 – Forme des équivalents affines de F.

— $F \in \mathrm{GL}(\mathbb{F}_2^n) \Leftrightarrow G \in \mathrm{GL}(\mathbb{F}_2^n)$.

Rappelons la définition d'un multiensemble : un multiensemble, noté entre accolades doubles, est une généralisation des ensembles dans laquelle une valeur peut apparaître plusieurs fois. Par exemple, $\{\{1, 2, 2, 2, 4, 8\}\}$ est un multiensemble dans lequel la valeur 2 a une multiplicité 3. Comme pour les ensembles, l'égalité entre multiensembles ne tient pas compte de l'ordre : $\{\{1, 2, 2, 2, 4, 8\}\} = \{\{8, 4, 2, 2, 2, 1\}\}$. Ainsi, $\{\{\mathrm{DDT}(F)\}\} = \{\{\mathrm{DDT}(G)\}\}$ signifie que, à réordonnancement près, les valeurs sont les mêmes dans les DDT de F et de G[8].

On remarquera que l'équivalence affine préserve plus que δ, $\mathcal{L}$, deg et la bijectivité : elle préserve aussi l'ensemble des valeurs présentes dans les DDT et LAT.

On peut vouloir s'intéresser à un cas d'équivalence moins contraignant, qui ne préserve pas la DDT et la LAT. C'est le principe de l'équivalence CCZ (Carlet, Charpin, Zinoviev [CCZ98]).

Définition 2.3 (Équivalence CCZ)**.** Soient F et G deux fonctions de $\mathbb{F}_2^n$ dans $\mathbb{F}_2^n$. F et G sont dites CCZ-équivalentes si[9]

$$\{(x, F(x)) \,|\, x \in \mathbb{F}_2^n\} \overset{\mathsf{Aff}}{\sim} \{(x, G(x)) \,|\, x \in \mathbb{F}_2^n\}\,,$$

ou entre d'autres termes, s'il existe L une permutation linéaire de $\mathbb{F}_2^n$ dans $\mathbb{F}_2^n$ telle que

$$\{(x, F(x)) \,|\, x \in \mathbb{F}_2^n\} = \{L(x, G(x)) \,|\, x \in \mathbb{F}_2^n\}\,.$$

Dans le cas d'un chiffrement, les couples $(x, F(x))$ sont des couples clair-chiffré. Les équivalents CCZ d'un chiffrement sont donc les autres chiffrements obtenus par transformation linéaire des couples clair-chiffré (ou des couples entrée-sortie si l'on n'a pas un chiffrement).

Propriété 2.2 (Équivalence CCZ)**.** *Si* $F \overset{CCZ}{\sim} G$*, alors :*
— $\delta(F) = \delta(G)$,
— $\mathcal{L}(F) = \mathcal{L}(G)$.

L'équivalence CCZ semble donc être la bonne notion pour étudier l'uniformité différentielle et la linéarité. Malheureusement, cette notion est plus complexe que l'équivalence linéaire, et donc il est difficile de prouver que deux fonctions sont équivalentes CCZ en général.

Remarquons en passant qu'il y a une inclusion de l'équivalence affine dans l'équivalence CCZ :

$$F \overset{\mathsf{Aff}}{\sim} G \Rightarrow F \overset{CCZ}{\sim} G\,.$$

L'utilisation de telles équivalences a un intérêt multiple :

8. A_1 correspond à un réordonnancement des colonnes, A_2 à un réordonnancement des lignes.
9. La notion d'équivalence affine entre deux ensembles A et B est à comprendre comme : toute fonction de B peut être obtenue comme une équivalente affine d'une fonction de A.

— à partir d'une fonction avec de bonnes propriétés cryptographiques, on peut en trouver d'autres,
— à partir d'une fonction sécurisée, on peut chercher un équivalent moins coûteux,
— à partir d'une fonction avec de bonnes propriétés, on peut chercher un équivalent plus simple à analyser (en particulier, un équivalent structuré),
— sur les chiffrements à flot, certains travaux ([CR16]) définissent des chiffrements équivalents, donnant la même suite chiffrante, et arrivent à ramener l'étude à un équivalent plus simple que le chiffrement d'origine afin d'attaquer le chiffrement. A priori, aucune attaque similaire n'existe sur les chiffrements par blocs, mais il n'est pas impossible qu'il existe une façon similaire d'exploiter l'équivalence entre chiffrements pour monter des attaques.

À l'inverse, on peut vouloir s'intéresser à un cas d'équivalence plus contraignant que l'équivalence affine. Un exemple d'une telle équivalence est l'équivalence par permutation des bits d'entrée et de sortie. Non seulement cette équivalence préserve l'uniformité différentielle, la linéarité et le degré algébrique, mais elle préserve aussi d'autres propriétés intéressantes : le nombre d'apparitions de l'uniformité différentielle dans chaque ligne de la DDT, le nombre d'apparitions de la linéarité (en valeur absolue) dans chaque ligne de la LAT, les facteurs de branchement différentiels et linéaires (utiles à la diffusion, cf. section 1.3.4) et le coût d'implémentation en portes logiques.

2.2 Boîtes-S à bas coût, une étude zoologique : Feistel et MISTY

Les paragraphes suivants détaillent les études et les résultats obtenus sur les schémas de Feistel et de type MISTY. Une comparaison avec les Papillons est donnée par la suite en section 2.4.3.

Il s'agit de travaux publiés à SAC 2015 [CDL16]. Pour les définitions, se référer à la section 1.3.

Notons que, jusqu'à récemment, la construction de boîtes-S structurées était délaissée, au profit de boîtes-S avec une représentation mathématique simple et à l'étude de boîtes-S offrant beaucoup de sécurité sans souci des coûts d'implémentation.

Tout comme pour les chiffrements, nous pouvons utiliser des structures comme les réseaux de Feistel afin de construire des boîtes-S de taille moyenne à partir de boîtes-S de taille plus petite. Mais contrairement au cas des chiffrements, il s'agit alors d'étudier la sécurité d'une boîte-S, qui ne dépend pas d'une clef secrète. Le modèle d'étude est donc fondamentalement différent de celui utilisé généralement pour l'étude de chiffrement : dans le cas des boîtes-S, nous nous intéresserons à un *modèle sans clef* (ce qui est généralement équivalent à avoir une clef fixée, on pourra donc parler de *modèle à clef fixée*). Pour plus de détails, se référer à la section 2.1.2.3.

2.2.1 Travaux antérieurs

Je compile ici un résumé de travaux intéressants et utiles à la recherche de bonnes boîtes-S. Il s'agit de recherches exhaustives sur des boîtes-S de petite taille.

Recherches exhaustives. Il existe des classes d'équivalence utiles pour étudier les boîtes-S (cf. section 2.1.3.2) en particulier l'équivalence affine (potentiellement étendue) et l'équivalence CCZ.

Grâce à ces équivalences, il est possible d'étudier de classifier toutes les boîtes-S de dimension petite. Ces travaux on été effectués dans plusieurs articles, notamment [BL08]

pour la classification des fonctions APN, [LP07a] pour la classification des boîtes-S bijectives les plus résistantes en dimension 4 et [Ull+11] pour la recherche de boîtes-S avec des circuits peu coûteux, et [Saa12] offre une étude plus fine des meilleurs résultats de [LP07a].

Sans rentrer dans tous les détails, je compile ici les résultats principaux de ces classifications :

- il n'y a pas de permutation APN en dimension 4,
- il y a 2 classes d'équivalence affine étendue de fonctions APN en dimension 4,
- il y a 1 classe d'équivalence CCZ de fonctions APN en dimension 4,
- il y a 5 classes d'équivalence affine étendue de fonctions APN en dimension 5,
- il y a 3 classe d'équivalence CCZ de fonctions APN en dimension 5,
- toute fonction APN en dimension 5 est équivalente CCZ à une fonction puissance (de la forme $x \mapsto x^e$),
- il y a au moins 14 classes d'équivalence CCZ de fonctions APN en dimension 6,
- il y a 1396032 permutations avec $\delta = 4$, $\mathcal{L} = 8$ en dimension 4, réparties en 16 classes d'équivalence affine étendue, parmi 302 classes d'équivalence affine étendue de permutations,
- la meilleure implémentation d'une permutation en dimension 4 avec $\delta = 4$, $\mathcal{L} = 8$ requiert 4 portes and ou or binaires et 4 portes xor.

Construction de boîtes-S à bas coût. Peu de travaux antérieurs s'étaient intéressés à pousser loin la réduction des coûts des boîtes-S. Sur le sujet des réseaux de Feistel, une brève étude préliminaire de la résistance de 3 tours de réseau de Feistel avait été proposée par Li et Wang [LW14], leurs résultats sont résumés ci-dessous.

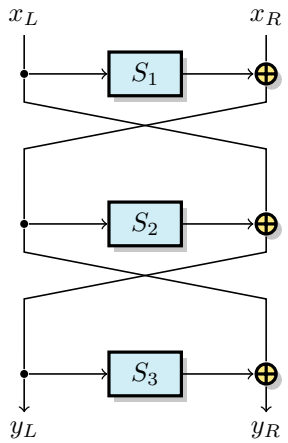


Figure 2.5 – Schéma de 3 tours d'un réseau de Feistel sans clef.

Théorème 2.2 (Sécurité de 3 tours de Feistel (Li et Wang [LW14]))**.** *Soit F la fonction obtenue par 3 tours d'un réseau de Feistel utilisant pour fonctions de tour les boîtes-S de n bits S_1, S_2 et S_3. Alors*

$$\delta(F) \geqslant 2\delta(S_2)\,.$$

De plus, si S_2 n'est pas une permutation, $\delta(F) \geqslant 2^{n+1}$.

Théorème 2.3 (Sécurité de 3 tours de Feistel sur 8 bits (Li et Wang [LW14]))**.** *Pour $n = 4$, F satisfait*

$$\delta(F) \geqslant 8\,.$$

et lorsque $\delta(F) = 8$, alors $\mathcal{L}(F) \geqslant 64$.

Li et Wang ne se sont pas contentés de bornes, mais ont exhibé un exemple précis de boîte-S construite à partir de 3 tours de réseau de Feistel atteignant les bornes $\delta = 8$, $\mathcal{L} = 64$.

2.2.2 Nos résultats

Dans notre article de 2015 publié à SAC, nous raffinons ces bornes et allons beaucoup plus loin dans l'analyse et jusqu'à des implémentations peu coûteuses de bons choix de S_1, S_2 et S_3 grâce à une représentation en circuit binaire. Les résultats sont donnés dans le théorème suivant, les preuves sont au paragraphe 2.2.3 et les travaux sur l'implémentation sont développés au paragraphe 2.4.1.

Théorème 2.4 (Sécurité de 3 tours de Feistel (SAC 2015 [CDL16]))**.** *Soit F la fonction obtenue par 3 tours d'un réseau de Feistel utilisant pour fonctions de tour les boîtes-S de n bits S_1, S_2 et S_3. Alors*
— *si S_2 n'est pas bijective, $\delta(F) \geqslant 2^{n+1}$,*
— *sinon, $\delta(F) \geqslant \max_{1\leqslant i\leqslant 3, j\neq i}\{\delta(S_i)\delta'_{\min}(S_j)\}$,*
où $\delta'_{\min}(S_j) = \delta_{\min}(S_j)$, sauf pour $j = 2$, où $\delta'_{\min}(S_2) = \delta_{\min}(S_2^{-1})$.
— *si S_2 n'est pas bijective, $\mathcal{L}(F) \geqslant \mathcal{L}(S_2)\max\left(\mathcal{L}_{min}(S_1),\ \mathcal{L}_{min}(S_3)\right)$,*
— *sinon, $\mathcal{L}(F) \geqslant \max_{1\leqslant i\leqslant 3, j\neq i}\{\mathcal{L}(S_i)\mathcal{L}'_{min}(S_j)\}$,*
où $\mathcal{L}'_{min}(S_j) = \mathcal{L}_{min}(S_j)$, sauf pour $j = 2$, où $\mathcal{L}'_{min}(S_2) = \mathcal{L}_{min}(S_2^{-1})$.

Dans le cas des réseaux de Feistel de 8 bits, nous obtenons les mêmes résultats que Li et Wang : $\delta(F) \geqslant 8$, $\mathcal{L}(F) \geqslant 64$.

Tout comme Li et Wang, nous pouvons exhiber de nombreux exemples de boîtes-S construites ainsi avec des paramètres de sécurité $\delta = 8$ et $\mathcal{L} = 64$ sur 8 bits. Par exemple,

$$S_1 = [0, 4, 4, 4, 13, 1, 7, 15, 4, 5, 2, 9, 13, 4, 5, 6]\,,$$

$$S_2 = [0, 1, 8, 9, 2, 14, 11, 7, 413, 15, 10, 6, 3, 12, 5]\,,$$

$$S_3 = [0, 2, 13, 6, 4, 14, 14, 6, 7, 14, 2, 2, 2, 3, 0, 3]$$

donne $\delta(F) = 8$ et $\mathcal{L}(F) = 64$.

Voici une liste de conditions nécessaires pour atteindre les bornes $\delta = 8$ et $\mathcal{L} = 64$:
— $\delta(S_1) = 2$ (en particulier, S_1 non-bijective[10]),
— $\delta(S_2) = 4$ et S_2 bijective,
— $\delta(S_3) = 2$ (en particulier, S_3 non-bijective[10]).

Ces conditions sont nécessaires pour qu'il n'y ait pas de caractéristique différentielle à travers les 3 tours de réseau de Feistel vérifiée pour strictement plus de 8 solutions.

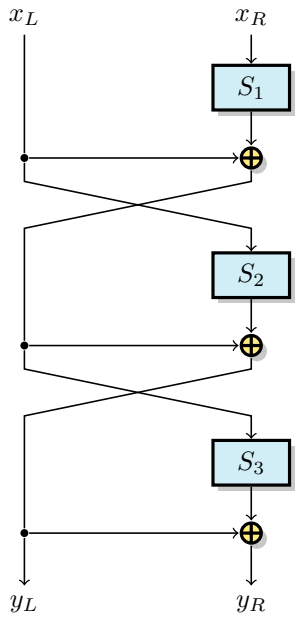
En revanche, il ne s'agit pas de conditions suffisantes : pour obtenir $\delta(F) = 8$, il faut que, pour toute caractéristique C à travers les 3 tours de réseau de Feistel vérifiée par 8 solutions, il n'existe aucune autre caractéristique non-nulle à travers ce réseau ayant les mêmes différences en entrée et en sortie que C. Ceci ne peut pas se décrire en considérant indépendamment S_1, S_2 et S_3 et nécessite d'étudier en détail les interactions entre ces 3 fonctions, ce qui rend la tâche très compliquée.

En pratique, quelques essais en faisant varier S_1, S_2 et S_3 qui vérifient les conditions suffisent à atteindre F avec $\delta(F) = 8$ et $\mathcal{L}(F) = 64$.

Les résultats sur 3 tours d'un réseau de type MISTY sont similaires.

Théorème 2.5 (Sécurité de 3 tours de MISTY (SAC 2015 [CDL16]))**.** *Soit F la fonction obtenue par 3 tours d'un réseau de type MISTY utilisant pour fonctions de tour les boîtes-S de n bits S_1, S_2 et S_3. Alors*
— *si S_1 n'est pas bijective, $\delta(F) \geqslant 2^{n+1}$,*

10. puisqu'il n'existe pas de bijection APN sur 4 bits, cf. section 1.3.6.4



— *sinon,* $\delta(F) \geqslant \max_{1\leqslant i\leqslant 3, j\neq i}\{\delta(S_i)\delta'_{\min}(S_j)\}$,
où $\delta'_{\min}(S_j) = \delta_{\min}(S_j)$, *sauf pour* $j = 1$, *où* $\delta'_{\min}(S_1) = \delta_{\min}(S_1^{-1})$.
— $\mathcal{L}(F) \geqslant \mathcal{L}(S_1)\max\left(\mathcal{L}(S_1)\mathcal{L}_{\min}(S_2), \mathcal{L}(S_2)\mathcal{L}_{\min}(S_1),\ \mathcal{L}(S_3)\mathcal{L}_{\min}(S_1)\right)$,
— *si* S_3 *est bijective,* $\mathcal{L}(F) \geqslant \mathcal{L}(S_1)\mathcal{L}_{\min}(S_3^{-1})$,
— *si* S_1 *est bijective,* $\mathcal{L}(F) \geqslant \mathcal{L}(S_3)\mathcal{L}_{\min}(S_2)$,
— *si* S_1 *et* S_3 *sont bijectives,* $\mathcal{L}(F) \geqslant \max_{1\leqslant i\leqslant 3, j\neq i}\{\mathcal{L}(S_i)\mathcal{L}'_{min}(S_j)\}$,
où $\mathcal{L}'_{min}(S_j) = \mathcal{L}_{min}(S_j)$, *sauf pour* $j = 3$, *où* $\mathcal{L}'_{min}(S_3) = \mathcal{L}_{min}(S_3^{-1})$.

Théorème 2.6 (Sécurité de 3 tours de MISTY sur 8 bits (SAC 2015 [CDL16]))**.** *Pour* $n = 4$, F *satisfait*

$$\delta(F) \geqslant 8\,,$$
$$\mathcal{L}(F) \geqslant 48\,.$$

et si $\delta(F) = 8$, *alors* $\mathcal{L}(F) \geqslant 64$.

De plus, si F *est une permutation, alors*

$$\delta(F) \geqslant 16\,.$$

On obtient les conditions nécessaires suivantes pour atteindre les bornes $\delta = 8$ et $\mathcal{L} = 64$:
— $\delta(S_1) = 4$ et S_1 bijective,
— $\delta(S_2) = 2$ (en particulier, S_2 non-bijective [11]),
— $\delta(S_3) = 2$ (en particulier, S_3 non-bijective[11]).

Rappelons que la boîte-S construite par 3 tours d'un réseau de type MISTY n'est une permutation que si S_1, S_2 et S_3 sont des permutations. Les conditions nécessaires impliquent donc que les bornes $\delta = 8$, $\mathcal{L} = 64$ ne sont atteintes que pour des boîtes-S non-bijectives, ce qui est désagréable et moins bon que le cas des réseaux de Feistel. Comme énoncé dans le théorème, dans le cas où le réseau de type MISTY est demandé bijectif, la meilleure borne atteignable est $\delta = 16$, $\mathcal{L} = 64$.

Il apparaît donc que, grâce à sa propriété de bijectivité intrinsèque, le réseau de Feistel est plus adapté pour construire des boîtes-S que le réseau de type MISTY (du moins tant qu'on préfère des boîtes-S bijectives).

11. puisqu'il n'existe pas de bijection APN sur 4 bits, cf. section 1.3.6.4

2.2.3 Feistel et MISTY : preuves

Que ce soit pour 3 tours de réseau de Feistel ou 3 tours de réseau de type MISTY, nous utiliserons la même approche.

L'élément essentiel de la preuve est de remarquer que quelques différentielles (resp. masques linéaires) sont toujours intéressantes pour un attaquant. Il s'agit des différentielles (resp. masques linéaires) qui donnent une différence (resp. un masque) nulle en entrée à l'une des 3 boîtes-S (cf. figures 2.6a, 2.6b, 2.6c, 2.7a, 2.7b et 2.7c pour Feistel et 2.8a, 2.8b, 2.8c, 2.9a, 2.9b, 2.9c pour MISTY).

Ainsi, en nous restreignant à ces quelques différentielles (resp. masques), nous pouvons aisément calculer leur nombre de « solutions » (il s'agit d'un abus de langage : j'appellerai « solution » d'une différentielle (a, b) pour la fonction F une solution x de l'équation différentielle $F(x) \oplus F(x \oplus a) = b$, et de même par extension pour le cas linéaire), qui implique des bornes générales sur la sécurité des réseaux Feistel et MISTY sur 3 tours.

Ensuite, en nous restreignant à des réseaux sur 8 bits, nous pouvons utiliser les propriétés de fonctions de 4 bits, qui sont étudiables presque exhaustivement sur une taille aussi petite, pour déduire des bornes précises sur ce cas particulier qui est essentiel en pratique.

Le cas des réseaux de 8 bits est offre un intérêt particulier, puisque dans ce cas, les bornes générales (pour tout n) sont atteintes : il existe des réseaux Feistel et MISTY qui atteignent les bornes générales.

Ceci signifie que nos bornes générales sont optimales, même si une instance spécifique de réseau Feistel ou MISTY peut avoir une uniformité différentielle (resp. linéarité) supérieure aux bornes.

Ainsi, les différentielles annulant l'entrée d'une boîte-S sont suffisantes pour obtenir une borne générique, ce qui simplifie énormément l'analyse et c'est cette propriété remarquable qui nous permet d'obtenir les bornes optimales pour un réseau de Feistel ou un réseau de type MISTY de 3 tours.

2.2.3.1 Uniformité différentielle de 3 tours de Feistel

Les différentielles annulant une boîte-S.

Puisque seules 2 boîtes-S sur 3 sont actives dans ces cas, calculer leur nombre de solution est plus simple que dans le cas de différentielles génériques.

En particulier, on remarque que, dès que la différentielle est fixée, *i.e.* dès que a, b et c sont fixées sur les schémas, il n'y a qu'une seule caractéristique (ou aucune) qui satisfait cette différentielle.

Théorème 2.7 (Différentielles annulant une boîte-S)**.** *Soit F la fonction obtenue par 3 tours d'un réseau de Feistel utilisant pour fonctions de tour les boîtes-S de n bits S_1, S_2 et S_3. Alors, pour tous a, b et c dans $\mathbb{F}_2^n$, on a :*

(i) $$\delta_F(0||a, b||c) = \delta_{S_2}(a, b) \times \delta_{S_3}(b, a \oplus c)\,;$$

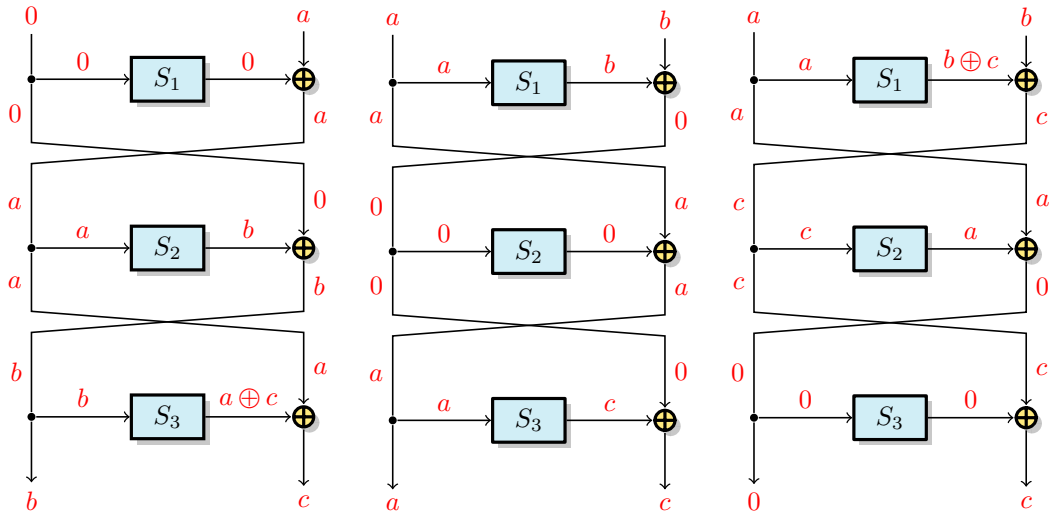
(ii) *Si S_2 est une permutation,*

$$\delta_F(a||b, a||c)| = \delta_{S_1}(a, b) \times \delta_{S_3}(a, c)\,;$$

(iii) $$\delta_F(a||b, 0||c) = \delta_{S_1}(a, b \oplus c) \times \delta_{S_2}(c, a)\,.$$

Démonstration. **(i)** $x = (x_L, x_R)$ satisfait $F(x_L||x_R) \oplus F(x_L||(x_R \oplus a)) = b||c$ si et seulement si

$$\begin{cases} S_2(S_1(x_L) \oplus x_R \oplus a) \oplus S_2(S_1(x_L) \oplus x_R) = b \\ S_3(y_R \oplus b) \oplus S_1(x_L) \oplus x_R \oplus a \oplus S_3(y_R) \oplus S_1(x_L) \oplus x_R = c \end{cases}$$



(a) Différentielles $(0||a, b||c)$ sur 3 tours d'un réseau de Feistel qui annulent la différence en entrée de S_1.

(b) Différentielles $(a||b, a||c)$ sur 3 tours d'un réseau de Feistel qui annulent la différence en entrée de S_2.

(c) Différentielles $(a||b, 0||c)$ sur 3 tours d'un réseau de Feistel qui annulent la différence en entrée de S_3.

Figure 2.6 – Propagation des différences dans 3 tours d'un réseau de Feistel pour les différentielles annulant les fonctions internes. Les valeurs en rouge sont des différences.

ou de façon équivalente,

$$S_1(x_L) \oplus x_R \in D_{S_2}(a \to b) \text{ et } y_R \in D_{S_3}(b \to a \oplus c) \ .$$

La fonction $(x_L, x_R) \mapsto ((S_1(x_L) \oplus x_R), y_R)$ est une permutation puisqu'il s'agit de 2 tours d'un réseau de Feistel. Ainsi, le nombre de $x = (x_L, x_R)$ qui satisfont la différentielle est $\delta_{S_2}(a \to b) \times \delta_{S_3}(b \to a \oplus c)$.

(ii) $x = (x_L, x_R)$ satisfait $F(x_L||x_R) \oplus F((x_L \oplus a)||(x_R \oplus b)) = c||a$ ssi

$$\begin{cases} S_2(S_1(x_L \oplus a) \oplus x_R \oplus b) \oplus x_L \oplus a \oplus S_2(S_1(x_L) \oplus x_R) \oplus x_L &= a \\ S_3(y_R \oplus a) \oplus S_1(x_L \oplus a) \oplus x_R \oplus b \oplus S_3(y_R) \oplus S_1(x_L) \oplus x_R &= c \end{cases}$$

La première équation correspond à

$$S_2\left(S_1(x_L \oplus a) \oplus x_R \oplus b\right) \oplus S_2\left(S_1(x_L) \oplus x_R\right) = 0 \ ,$$

qui équivaut à

$$S_1(x_L \oplus a) \oplus x_R \oplus b = S_1(x_L) \oplus x_R$$

si S_2 est une permutation. Ainsi, (x_L, x_R) satisfait la différentielle ssi

$$x_L \in D_{S_1}(a \to b) \text{ et } y_R \in D_{S_3}(a \to c) \ .$$

Pour tout $x_L \in D_{S_1}(a \to b)$ fixé, il y a $\delta_{S_3}(a, c)$ valeurs de $x_R = S_2^{-1}(x_L \oplus y_R) \oplus S_1(x_L)$ qui donnent une sortie valide.

(iii) $x = (x_L, x_R)$ satisfait $F(x_L||x_R) \oplus F((x_L \oplus a)||(x_R \oplus b)) = c||0$ ssi

$$\begin{cases} S_2(S_1(x_L \oplus a) \oplus x_R \oplus b) \oplus x_L \oplus a \oplus S_2(S_1(x_L) \oplus x_R) \oplus x_L &= 0 \\ S_3(y_R) \oplus S_1(x_L \oplus a) \oplus x_R \oplus b \oplus S_3(y_R) \oplus S_1(x_L) \oplus x_R &= c \end{cases}$$

La deuxième équation est équivalente à

$$x_L \in D_{S_1}(a \to b \oplus c)$$

et la première correspond à

$$S_1(x_L) \oplus x_R \in D_{S_2}(c \to a) \ . \qquad \square$$

Borne générale.

De ces différentielles particulières, nous déduisons des bornes inférieures sur l'uniformité différentielle de 3 tours de réseau de Feistel (cf. théorème 2.4).

Théorème 2.8. *Soit F la fonction obtenue par 3 tours d'un réseau de Feistel utilisant pour fonctions de tour les boîtes-S de n bits S_1, S_2 et S_3. Alors*

$$\delta(F) \geqslant \delta(S_2) \max\left(\delta_{\min}(S_1), \delta_{\min}(S_3)\right) \ .$$

De plus,

— *si S_2 est une permutation,*

$$\delta(F) \geqslant \max_{i\neq 2, j\neq 2} \left(\delta(S_i)\delta_{\min}(S_j),\ \delta(S_i)\delta_{\min}(S_2^{-1})\right) \ ,$$

— *si S_2 n'est pas une permutation, $\delta(F) \geqslant 2^{n+1}$.*

Démonstration.
Le résultat lorsque S_2 n'est pas une permutation a été prouvé dans [LW14]. Du premier item du théorème 2.7, nous déduisons que

$$\delta(F) \geqslant \delta(S_2)\delta_{\min}(S_3) \text{ et } \delta(F) \geqslant \delta(S_3)\delta_{\min}(S_2^{-2})$$

où la seconde inégalité requiert que S_2 soit une permutation. En effet, le premier résultat est obtenu en choisissant (a, b) tel que $\delta_{S_2}(a, b) = \delta(S_2)$ et en prenant le maximum sur tous les choix de c. Lorsqu'on choisit $(a \oplus c, b)$ tel que $\delta_{S_3}(b, a \oplus c) = \delta(S_3)$ et qu'on prend le maximum sur tous les choix de a, on obtient le second résultat en utilisant que

$$\max_{\alpha\neq 0} \delta_{S_2}(\alpha, \beta) = \max_{\alpha\neq 0} \delta_{S_2^{-1}}(\beta, \alpha) \geqslant \delta_{\min}(S_2^{-1})$$

lorsque S_2 est une permutation.

Le deuxième item du théorème 2.7, qui tient lorsque S_2 est une permutation, implique

$$\delta(F) \geqslant \delta(S_1)\delta_{\min}(S_3) \text{ et } \delta(F) \geqslant \delta(S_3)\delta_{\min}(S_1) \ .$$

La première inégalité est obtenue en choisissant (a, b) tel que $\delta_{S_1}(a, b) = \delta(S_1)$ et en prenant le maximum sur tous les choix de c, et la seconde en choisissant (a, c) tel que $\delta_{S_3}(a, c) = \delta(S_3)$ puis en prenant le maximum sur tous les choix de b.

Le troisième item du théorème 2.7 implique

$$\delta(F) \geqslant \delta(S_1)\delta_{\min}(S_2^{-1}) \text{ et } \delta_F \geqslant \delta(S_2)\delta_{\min}(S_1) \ .$$

La première inégalité est obtenue en choisissant $(a, b\oplus c)$ tel que $\delta_{S_1}(a, b\oplus c) = \delta(S_1)$ et en prenant le maximum sur tous les choix de c lorsque S_2 est une permutation, et la seconde en choisissant (a, c) tel que $\delta_{S_2}(c, a) = \delta(S_2)$ puis en prenant le maximum sur tous les choix de b. $\square$

Application à $n = 4$.

De même que Li et Wang dans [LW14], nous pouvons déduire des bornes plus précises dans le cas $n = 4$, c'est-à-dire lorsque le réseau de Feistel opère sur 8 bits.

Les bornes obtenues sont les mêmes, à savoir

Proposition 2.1. *Soit F la fonction obtenue par 3 tours d'un réseau de Feistel utilisant pour fonctions de tour les boîtes-S de 4 bits S_1, S_2 et S_3. Alors*

$$\delta(F) \geqslant 8 .$$

Démonstration.
Il s'agit d'une conséquence du théorème 2.7.

Si S_2 n'est pas une permutation, alors la borne générale donne $\delta(F) \geqslant 2^5 = 32$.

Si S_2 est une permutation, on sait qu'il n'existe pas de permutation APN sur 4 bits, donc $\delta(S_2) \geqslant 4$, et par conséquent $\delta(F) \geqslant \delta(S_2)\delta_{\min}(S_1) \geqslant 4 \times 2 = 8$. □

Contrairement à Li et Wang, nous pouvons déduire des conditions nécessaires pour qu'un réseau de Feistel sur 8 bits F atteigne $\delta(F) = 8$.

La partie la plus compliquée de la preuve est détaillée dans le lemme 2.1, qui s'appuie sur les deux propriétés suivantes sur des boîtes-S de 4 bits suivantes.

Propriété 2.3. *Soit S une fonction de 4 bits telle que $\delta(S) \geqslant 4$.*

Alors il existe a et c non-nuls dans $\mathbb{F}_2^4$ et d_1, d_2 et d_3 dans $\mathbb{F}_2^4$ tels que :

$$\delta_S(a, d_1) \geqslant 4 ,$$

$$\delta_S(c, d_2) \geqslant 4 ,$$

$$\delta_S(a \oplus c, d_3) \geqslant 4 .$$

En d'autres termes, il existe au moins trois lignes de la table des différences de S contenant au moins une valeur supérieure ou égale à 4, et les différences d'entrée correspondant à ces trois lignes forment un ensemble stable par xor.

Démonstration.
Considérons un couple de différences non-nul (a, b) tel que $\delta_S(a, b) \geqslant 4$. Alors $D_S(a \to b)$ contient un sous-espace affine de dimension 2 de $\mathbb{F}_2^4$. En effet, si on choisit $x \in D_S(a \to b)$, il existe $c \in \mathbb{F}_2^4 \backslash \{0\}$, $c \neq a$, tel que $D_S(a \to b) \supseteq \{x, x \oplus a, x \oplus c, x \oplus c \oplus a\}$.

D'où nous déduisons que

$$S(x) \oplus S(x \oplus c) = S(x \oplus a) \oplus S((x \oplus a) \oplus c)$$

et

$$S(x) \oplus S(x \oplus (c \oplus a)) = S(x \oplus a) \oplus S((x \oplus a) \oplus (c \oplus a))$$

Ainsi les trois lignes définies par a, c et $a \oplus c$ de la table des différences de S contiennent une valeur supérieure ou égale à 4. □

Propriété 2.4. *Soit S une permutation de 4 bits avec $\delta(S) = 4$. Alors pour tous a et c non-nuls dans $\mathbb{F}_2^4$, au moins une colonne paramétrée par une différence dans $\{a, c, a \oplus c\}$ contient un 4.*

Démonstration.
Soit $\mathcal{C}(S)$ l'ensemble des colonnes de la table des différences d'une boîte-S ne contenant que des 0s et des 2s :

$$\mathcal{C}(S) = \{b \in \mathbb{F}_2^4 \backslash \{0\} \ : \delta_S(a, b) \leqslant 2, \forall a \neq 0\} .$$

Parmi toutes les classes d'équivalence de permutations de 4 bits S_2 avec $\delta(S) = 4$, seules 12 satisfont $\#\mathcal{C}(S) \geqslant 3$. De plus, 3 d'entre elles ont leur ensemble $\mathcal{C}(S)$ inclus dans l'ensemble $\mathcal{C}(S')$ d'une fonction dans une autre classe, aussi nous les ignorerons.

Les valeurs de $\mathcal{C}$ pour les 9 classes restantes sont listées ci-dessous.

Représentant de classe	$\mathcal{C}$
[0, 1, 2, 3, 4, 6, 9, A, 8, C, 5, D, B, E, F, 7]	$\{4, 5, 7, 9, 15\}$
[0, 1, 2, 3, 4, 6, 9, C, 8, 5, B, F, E, D, 7, A]	$\{4, 6, 7, 11, 14\}$
[0, 1, 2, 3, 4, 6, 9, C, 8, 5, F, D, B, 7, A, E]	$\{4, 10, 12, 13, 15\}$
[0, 1, 2, 3, 4, 6, 9, A, 8, B, C, E, F, 7, 5, D]	$\{4, 5, 11, 12, 13\}$
[0, 1, 2, 3, 4, 6, 9, A, 8, 5, C, F, D, B, E, 7]	$\{4, 9, 11, 14\}$
[0, 1, 2, 3, 4, 6, 9, A, 8, C, 5, D, 7, E, F, B]	$\{5, 9, 11, 15\}$
[0, 1, 2, 3, 4, 6, 9, A, 8, C, B, D, 5, F, E, 7]	$\{5, 7, 14, 15\}$
[0, 1, 2, 3, 4, 6, 9, C, 8, 5, D, A, E, 7, B, F]	$\{10, 14, 15\}$
[0, 1, 2, 3, 4, 6, 9, A, 8, B, C, E, 5, D, F, 7]	$\{6, 7, 11, 14, 15\}$

Nous pouvons vérifier qu'aucun de ces ensembles $\mathcal{C}(S)$ ne contient de sous-ensemble de 3 éléments stable par addition (xor). Qui plus est, cette propriété est invariante par transformation affine. En effet, pour $S' = A_2 \circ S \circ A_1$, nous avons

$$\delta_{S'}(a, b) = \delta_S(L_1(a), L_2^{-1}(b))$$

où L_1 et L_2 sont les parties linéaires de A_1 et A_2. Il s'ensuit que $\mathcal{C}(S') = L_2(\mathcal{C}(S))$.

Ainsi, parmi 3 éléments non-nuls de $\mathbb{F}_2^4$ stables par xor, au moins l'un des 3 définit une colonne de S contenant un 4. □

Lemme 2.1. *Soit F la fonction obtenue par 3 tours d'un réseau de Feistel utilisant pour fonctions de tour les boîtes-S de 4 bits S_1, S_2 et S_3. Si S_2 est une permutation avec $\delta(S_2) = 4$, alors si $\delta(S_1) \geqslant 4$ ou $\delta(S_3) \geqslant 4$, $\delta(F) \geqslant 16$.*

Démonstration.
Ici, S_1 et S_3 jouent un rôle symétrique. Nous supposerons que c'est S_3 qui est telle que $\delta(S_3) \geqslant 4$.

D'après la Propriété 2.3, il existe au moins trois lignes a, c et $a \oplus c$ de la table des différences de S_3 contenant au moins une valeur supérieure ou égale à 4.

Du premier item du théorème 2.7, nous déduisons que, pour tout $\beta \in \{a, c, a \oplus c\}$, nous pouvons choisir un γ tel que, pour tout α,

$$\delta_F(0\|\alpha, \beta\|\gamma) = \delta_{S_2}(\alpha, \beta) \times \delta_{S_3}(\beta, \alpha \oplus \gamma) \geqslant 4\delta_{S_2}(\alpha, \beta) \ .$$

De ce fait, $\delta(F) \geqslant 16$ à moins que les trois colonnes de la table des différences de S_2 définies par une différence dans $\{a, c, a \oplus c\}$ ne contiennent pas de 4.

Or d'après la Propriété 2.4, la table des différences de S_2 a au moins une colonne parmi $\{a, c, a \oplus c\}$ qui contient un 4. Choisir cette colonne donne donc une différentielle de F ayant au moins 16 solutions. □

Théorème 2.9. *Soit F la fonction obtenue par 3 tours d'un réseau de Feistel utilisant pour fonctions de tour les boîtes-S de 4 bits S_1, S_2 et S_3. Alors les conditions suivantes sont nécessaires pour avoir $\delta(F) = 8$:*

- *S_1 APN (donc pas une permutation),*
- *S_2 une permutation avec $\delta(S_2) = 4$,*
- *S_3 APN (donc pas une permutation).*

Sinon, $\delta(F) \geqslant 12$.

Démonstration.
D'après le théorème 2.7, on a besoin que S_2 soit une permutation, et si $\delta(S_2) \geqslant 6$, $\delta(F) \geqslant 12$, donc il faut que $\delta(S_2) \leqslant 4$. Comme il n'existe pas de permutation APN sur 4 bits, il faut donc $\delta(S_2) = 4$.

De plus, d'après le lemme 2.1, si S_1 ou S_3 n'est pas APN, alors $\delta(F) \geqslant 16$.

D'où le résultat. □

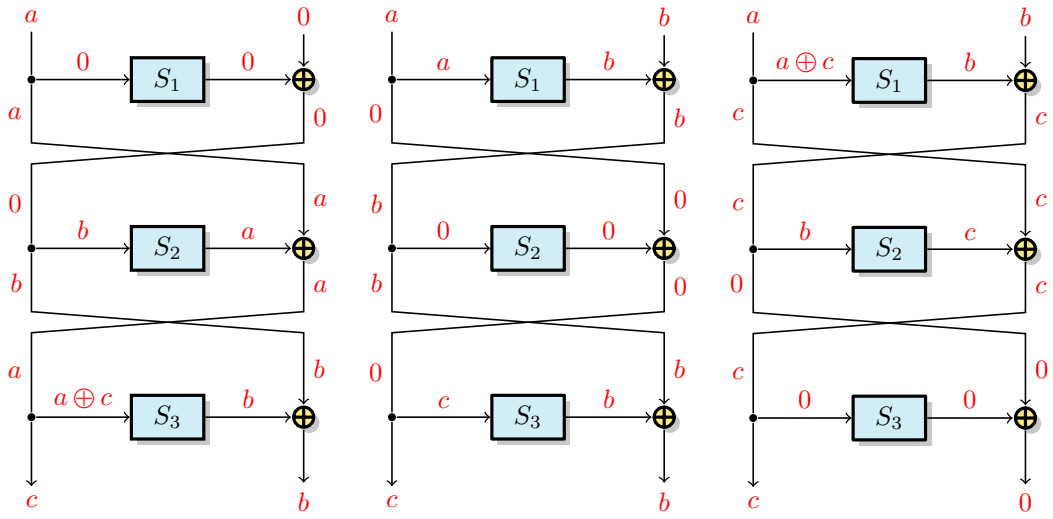
Remarque 2.2 (Comprendre le résultat sur 8 bits.). Il faut comprendre que le fait que la meilleure uniformité différentielle atteignable pour 8 bits soit 8 provient clairement de l'absence de permutations APN sur 4 bits. Pour d'autres tailles, il est tout à fait envisageable d'obtenir une uniformité différentielle égale à 4. Des exemples de ce type apparaissent dans les travaux sur les Papillons 2.3, mais avaient déjà été obtenus par Li et Wang dans [LW14] pour toute taille de la forme $2n$, n impair.

2.2.3.2 Linéarité de 3 tours de Feistel

L'étude de la linéarité est similaire à celle de l'uniformité différentielle.

Ici encore, on utilise le fait que certains masques sont optimaux pour l'attaquant dans le cas général.

Les masques annulant une boîte-S.



(a) Masques $(a||0, c||b)$ sur 3 tours d'un réseau de Feistel qui annulent le masque en entrée de S_1.

(b) Masques $(a||b, c||b)$ sur 3 tours d'un réseau de Feistel qui annulent le masque en entrée de S_2.

(c) Masques $(a||b, c||0)$ sur 3 tours d'un réseau de Feistel qui annulent le masque en entrée de S_3.

Figure 2.7 – Propagation des masques linéaires dans 3 tours d'un réseau de Feistel pour les masques annulant les fonctions internes. Les valeurs en rouge sont des masques linéaires.

Puisque seules 2 boîtes-S sur 3 sont actives dans le cas des masques annulant une boîte-S, les calculs sont simples.

Théorème 2.10. *Soit F la fonction obtenue par 3 tours d'un réseau de Feistel utilisant pour fonctions de tour les boîtes-S de n bits S_1, S_2 et S_3. Alors, pour tous a, b et c dans $\mathbb{F}_2^n$, on a :*

(i) $\hat{F}(a||b,c||0) = \hat{S}_1(a \oplus c, b)\hat{S}_2(b,c)$

(ii) $\hat{F}(a||0,c||b) = \hat{S}_2(b,a)\hat{S}_3(a \oplus c, b)$

(iii) *Si S_2 est une permutation,* $\hat{F}(a||b,c||b) = \hat{S}_1(a,b)\hat{S}_3(c,b)$

Démonstration.

(i)
$$\begin{aligned}\hat{F}(a||b,c||0) &= \sum_{(x_R,x_L)\in(\mathbb{F}_2^n)^2} (-1)^{c\cdot x_L + c\cdot S_2(S_1(x_L)\oplus x_R)\oplus a\cdot x_L \oplus b\cdot x_R} \\ &= \sum_{x_L\in\mathbb{F}_2^n} (-1)^{(a\oplus c)\cdot x_L + b\cdot S_1(x_L)} \sum_{z\in\mathbb{F}_2^n} (-1)^{c\cdot S_2(z)+b\cdot z} \\ &= \hat{S}_1(a\oplus c, b)\,\hat{S}_2(b,c)\end{aligned}$$

où on fixe $z = S_1(x_L) + x_R$ et on exploite le fait que, pour tout x_L fixé, $x_R \mapsto z$ est une permutation.

(ii)
$$\begin{aligned}\hat{F}(a||0,c||b) &= \sum_{(x_R,x_L)\in(\mathbb{F}_2^n)^2} (-1)^{c\cdot y_R + b\cdot S_3(y_R) + b\cdot(S_1(x_L)\oplus x_R)\oplus a\cdot x_L} \\ &= \sum_{(y_R,z)\in(\mathbb{F}_2^n)^2} (-1)^{c\cdot y_R + b\cdot S_3(y_R) + b\cdot z + a\cdot(y_R + S_2(z))} \\ &= \hat{S}_2(b,a)\,\hat{S}_3(a\oplus c, b)\end{aligned}$$

en remarquant que la fonction $(x_L, x_R) \mapsto (y_R, z)$ est une permutation puisqu'il s'agit de 2 tours d'un réseau de Feistel.

(iii)
$$\begin{aligned}\hat{F}(a||b,c||b) &= \sum_{(x_R,x_L)\in(\mathbb{F}_2^n)^2} (-1)^{c\cdot y_R + b\cdot S_3(y_R) + b\cdot x_R + b\cdot S_1(x_L)\oplus a\cdot x_L + b\cdot x_R} \\ &= \sum_{x_L\in\mathbb{F}_2^n} (-1)^{b\cdot S_1(x_L)\oplus a\cdot x_L} \sum_{y\in\mathbb{F}_2^n} (-1)^{c\cdot y + b\cdot S_3(y)} \\ &= \hat{S}_1(a,b)\,\hat{S}_3(c,b)\end{aligned}$$

en utilisant que $x_R \mapsto y = x_L \oplus S_2(S_1(x_L)\oplus x_R)$ est une permutation pour tout x_L fixé lorsque S_2 est une permutation.

□

Borne générale.

Nous en déduisons les bornes inférieures suivantes sur la linéarité de 3 tours d'un réseau de Feistel.

Théorème 2.11. *Soit F la fonction obtenue par 3 tours d'un réseau de Feistel utilisant pour fonctions de tour les boîtes-S de n bits S_1, S_2 et S_3. Alors,*

$$\mathcal{L}(F) \geqslant \mathcal{L}(S_2)\max\left(\mathcal{L}_{\min}(S_1),\ \mathcal{L}_{\min}(S_3)\right)\ .$$

De plus, si S_2 est une permutation,

$$\mathcal{L}(F) \geqslant \mathcal{L}(S_1)\max\left(\mathcal{L}_{\min}(S_2^{-1}),\ \mathcal{L}_{\min}(S_3)\right)$$

$$\text{et } \mathcal{L}(F) \geqslant \mathcal{L}(S_3)\max\left(\mathcal{L}_{\min}(S_1),\ \mathcal{L}_{\min}(S_2^{-1})\right)\ .$$

Démonstration.
Du premier item du théorème 2.10, nous déduisons que

$$\mathcal{L}(F) \geqslant \mathcal{L}(S_1)\mathcal{L}_{\min}(S_2^{-1}) \text{ et } \mathcal{L}(F) \geqslant \mathcal{L}(S_2)\mathcal{L}_{\min}(S_1)$$

où la première inégalité tient si S_2 est une permutation.

Le deuxième item du théorème 2.10 donne

$$\mathcal{L}(F) \geqslant \mathcal{L}(S_2)\mathcal{L}_{\min}(S_3) \text{ et } \mathcal{L}(F) \geqslant \mathcal{L}(S_3)\mathcal{L}_{\min}(S_2^{-1})$$

où la deuxième inégalité est valide si S_2 est une permutation.

Du troisième item du théorème 2.10, on obtient que, lorsque S_2 est une permutation

$$\mathcal{L}(F) \geqslant \mathcal{L}(S_1)\mathcal{L}_{\min}(S_3) \text{ et } \mathcal{L}(F) \geqslant \mathcal{L}(S_3)\mathcal{L}_{\min}(S_1) \ .$$

□

Application à $n = 4$.

À nouveau nous obtenons les mêmes bornes que Li et Wang dans le cas où $n = 4$.

Nous utilisons l'observation suivante :

Lemme 2.2. *Pour toute permutation de 4 bits S, la table des biais linéaires de S contient au moins une valeur supérieure ou égale à 8 dans toute ligne et toute colonne, c'est-à-dire que $\mathcal{L}_{min}(S) \geqslant 8$.*

Démonstration.
Ce résultat est obtenu par une recherche exhaustive sur les classes d'équivalence affine. Nous avons examiné les 302 représentants, et nous avons vérifié que le résultat est vrai pour tous. □

Proposition 2.2. *Soit F la fonction obtenue par 3 tours d'un réseau de Feistel utilisant pour fonctions de tour les boîtes-S de n bits S_1, S_2 et S_3. Si au moins une parmi S_1, S_2 et S_3 est une permutation, alors*

$$\mathcal{L}(F) \geqslant 64 \ .$$

Notablement, si $\mathcal{L}(F) < 64$, alors $\delta(F) \geqslant 32$.

Démonstration.
Il s'agit d'une conséquence directe du théorème 2.11 en utilisant le lemme 2.2.

En effet, on rappelle que d'après le théorème 2.1, lorsque S_2 n'est pas une permutation, $\delta(F) \geqslant 32$. □

Bien que cette proposition implique que le cas $\mathcal{L}(F) < 64$ a peu d'intérêt, nous pouvons prouver que de manière générale, $\mathcal{L}(F) \geqslant 48$.

Nous utiliserons les lemmes suivants qui permettent de prouver que toute fonction de 4 bits a toujours au moins une valeur supérieure ou égale à 6 dans chacune des colonnes de sa table des biais linéaires, c'est-à-dire que $\mathcal{L}_{\min}(S) \geqslant 6$ pour toute fonction de 4 bits S.

Lemme 2.3. *Les seules fonctions de 4 bits S avec $|\hat{S}(0,c)| < 4$ pour tout c non-nul sont les permutations, et les fonctions dont l'image est de taille 15.*

Démonstration.
La propriété $\forall c \neq 0$: $|\hat{S}(0,c)| < 4$ dépend uniquement des ensembles d'images de S, avec multiplicité. Ce résultat est obtenu par une recherche exhaustive sur les classes de 16 éléments dans $\{0,\ldots,15\}$. Nous avons testé la propriété sur toutes les 300540195 classes ; seule la classe avec 16 valeurs distinctes et les 240 classes avec 15 valeurs distinctes la satisfont. □

Lemme 2.4. *Pour toute fonction de 4 bits S dont l'image est de taille 15 ou 16, la table des biais linéaires de S contient au moins une valeur supérieure ou égale à 6 dans chaque colonne (*i.e. $\forall c \neq 0 \colon \exists a, |\hat{S}(a,c)| \geqslant 6$*).*

Démonstration.
La propriété est invariante par équivalence affine. Ainsi, nous obtenons le résultat par une recherche exhaustive sur les classes d'équivalence affine de S dont l'image est de taille 15 ou 16. D'après la classification des permutations de 4 bits [De 07], il y a 302 permutations P_k telles que toute permutation de 4 bits P puisse être décomposée en $P = \beta \circ P_k \circ \alpha$, avec α, β des permutations affines.

Soit S une fonction de 4 bits dont l'image est de taille 15. Nous noterons $\pi_{i,j}$ la projection $i \mapsto j$; $x \neq i \mapsto x$. Il existe une permutation P et une projection $\pi_{i,j}$ telles que $S = \pi_{i,j} \circ P$. En utilisant cette décomposition $P = \beta \circ P_k \circ \alpha$, nous démontrons que S est équivalente affinement à une fonction $\pi_{a,b} \circ P_k$:

$$S = \pi_{i,j} \circ \beta \circ P_k \circ \alpha = \beta \circ \pi_{\beta^{-1}(i),\beta^{-1}(j)} \circ P_k \circ \alpha.$$

Nous avons testé la propriété sur toutes les 302 permutations P_k et toutes les 72480 fonctions $\pi_{a,b} \circ P_k$. □

Proposition 2.3. *Soit F la fonction obtenue par 3 tours d'un réseau de Feistel utilisant pour fonctions de tour les boîtes-S de n bits S_1, S_2 et S_3. Alors $\mathcal{L}(F) \geqslant 48$.*

Démonstration.
La proposition 2.2 montre que le résultat est vrai si S_2 est une permutation. Supposons désormais que S_2 n'est pas une permutation, i.e., il existe un masque non-nul c tel que $\hat{S_2}(0,c) > 0$. D'après le premier item du théorème 2.10, nous obtenons que

$$\hat{F}(c||0,0||c) = \hat{S_1}(0,0)\hat{S_2}(0,c) = 16\hat{S_2}(0,c)\ .$$

Le résultat est vrai s'il existe un c non-nul tel que $\hat{S_2}(0,c) \geqslant 4$. Dans le cas contraire, nous déduisons des lemmes 2.3 et 2.4 que

$$\mathcal{L}(F) \geqslant 48\ .$$ □

Nous conjecturons qu'un réseau de Feistel avec des fonctions internes sur 4 bits satisfait même $\mathcal{L}(F) \geqslant 64$, mais ce résultat semble difficile à prouver sans une classification plus précise des fonctions de 4 bits.

2.2.3.3 Uniformité différentielle de 3 tours de MISTY

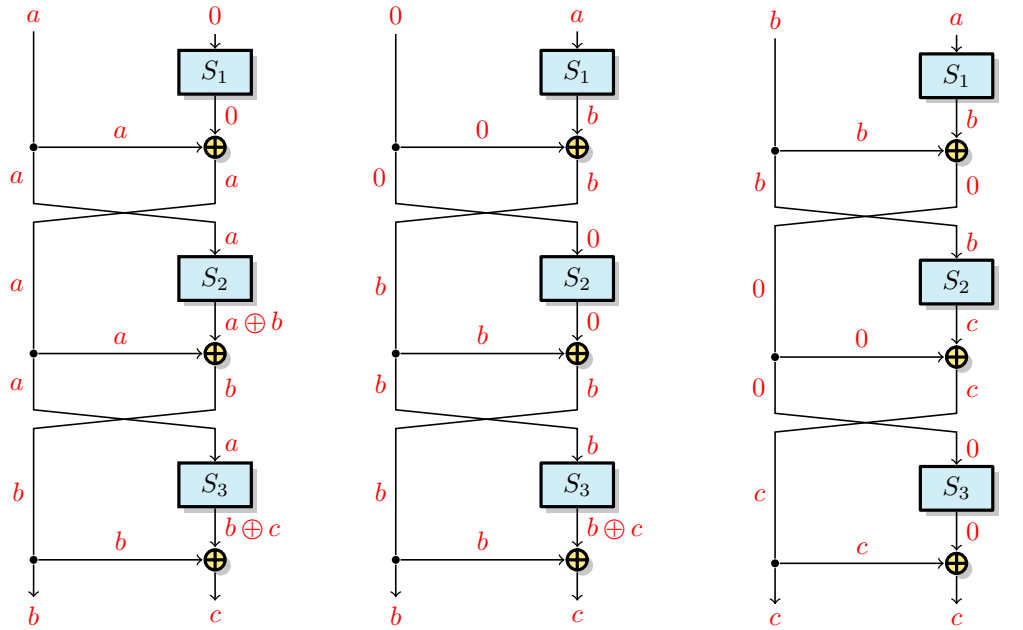
La seule différence importante entre la sécurité d'un réseau de type MISTY et celle d'un réseau de Feistel vient du fait qu'un réseau de Feistel est une permutation quels que soient les choix de S_1, S_2 et S_3, mais pour un réseau de type MISTY, on n'obtient une permutation que si S_1, S_2 et S_3 sont elles-mêmes des permutations.

Ainsi, toute l'analyse de sécurité se déroulera de façon identique au cas des réseaux de Feistel, avec simplement une distinction des cas qui génèrent des permutations et de ceux qui n'en génèrent pas.

Les différentielles annulant une boîte-S.

Ici encore, on a des différentielles qui annulent une boîte-S parmi S_1, S_2 et S_3 qui sont et qui permettent d'obtenir une borne générique.

Théorème 2.12 (Différentielles optimales)**.** *Soit F la fonction obtenue par 3 tours d'un réseau de type MISTY utilisant pour fonctions de tour les boîtes-S de n bits S_1, S_2 et S_3. Alors, pour tous a, b et c dans $\mathbb{F}_2^n$, on a :*



(a) Différentielles $(a||0, b||c)$ sur 3 tours d'un réseau de type MISTY qui annulent la différence en entrée de S_1.

(b) Différentielles $(0||a, b||c)$ sur 3 tours d'un réseau de type MISTY qui annulent la différence en entrée de S_2.

(c) Différentielles $(b||a, c||c)$ sur 3 tours d'un réseau de type MISTY qui annulent la différence en entrée de S_3.

Figure 2.8 – Propagation des différences dans 3 tours d'un réseau de type MISTY pour les différentielles annulant les fonctions internes. Les valeurs en rouge sont des différences.

(i) $$\delta_F(0\|a, b\|c) = \delta_{S_1}(a, c) \times \delta_{S_3}(c, b \oplus c)\,;$$

(ii) *Si S_1 est une permutation,*

$$\delta_F(a\|0, b\|c) = \delta_{S_2}(a, a \oplus c) \times \delta_{S_3}(a, b \oplus c)\,;$$

(iii) $\delta_{S_1}(a, b) \times \delta_{S_2}(b, c) \leqslant \delta_F(b\|a, c\|c) \leqslant \sum_{d \in \mathbb{F}_2^n} \delta_{S_1}(a, b \oplus d) \times \delta_{S_2}(b, c \oplus d) \times \gamma_{S_3}(d)$

où $\gamma_{S_3}(d)$ vaut 0 si $\delta_{S_3}(d, 0) = 0$ et 1 sinon.
Notablement, si S_3 est une permutation,

$$\delta_F(b\|a, c\|c) = \delta_{S_1}(a, b) \times \delta_{S_2}(b, c)\,.$$

Démonstration.
Soit x l'entrée d'un réseau de type MISTY sur 3 tours, x_L et x_R ses moitiés droite et gauche respectivement. Les trois résultats que nous prouvons correspondent aux configurations illustrées en figures 2.8a, 2.8b et 2.8c.

(i) $x = (x_L, x_R)$ satisfait $F(x_L\|x_R) \oplus F(x_L\|(x_R \oplus a)) = b\|c$ si et seulement si

$$\begin{cases} S_3(S_1(x_R) \oplus x_L) \oplus S_3(S_1(x_R \oplus a) \oplus x_L) = b \oplus c, \\ S_2(x_L) \oplus S_1(x_R) \oplus x_L \oplus S_2(x_L) \oplus S_1(x_R \oplus a) \oplus x_L = c \end{cases}$$
$$\Leftrightarrow \begin{cases} S_3(S_1(x_R) \oplus x_L) \oplus S_3(S_1(x_R \oplus a) \oplus x_L) = b \oplus c, \\ S_1(x_R) \oplus S_1(x_R \oplus a) = c \end{cases}$$

ou de façon équivalente

$$x_R \in D_{S_1}(a \to c) \text{ et } x_L \in S_1(x_R) \oplus D_{S_3}(c \to b \oplus c)\,.$$

D'où l'on déduit qu'il y a exactement $\delta_{S_1}(a,c)$ valeurs de x_R, et pour chacune de celles-ci, $\delta_{S_3}(c, b \oplus c)$ valeurs de x_L, telles que x vérifie la différentielle.

(ii) $x = (x_L, x_R)$ satisfait $F(x_L \| x_R) \oplus F((x_L \oplus a) \| x_R) = b \| c$ si et seulement si

$$\begin{cases} S_3(S_1(x_R) \oplus x_L) \oplus S_3(S_1(x_R) \oplus x_L \oplus a) = b \oplus c, \\ S_2(x_L) \oplus S_1(x_R) \oplus x_L \oplus S_2(x_L \oplus a) \oplus S_1(x_R) \oplus x_L \oplus a = c \end{cases}$$
$$\Leftrightarrow \begin{cases} S_1(x_R) \oplus x_L \in D_{S_3}(a \to b \oplus c), \\ S_2(x_L) \oplus S_2(x_L \oplus a) = a \oplus c \end{cases}$$

ou de façon équivalente,

$$x_L \in D_{S_2}(a \to a \oplus c) \text{ et } S_1(x_R) \in x_L \oplus D_{S_3}(a \to b \oplus c) .$$

Si S_1 est bijective, pour tout x_L fixé, chacune des $\delta_{S_3}(a, b \oplus c)$ valeurs définies par la seconde condition détermine une unique valeur pour x_R. Ainsi, le nombre de (x_L, x_R) satisfaisant la différentielle est exactement $\delta_{S_2}(a, a \oplus c) \times \delta_{S_3}(a, b \oplus c)$.

(iii) (x_L, x_R) satisfait $F(x_L \| x_R) \oplus F((x_L \oplus b) \| (x_R \oplus a)) = c \| c$ si et seulement si

$$\begin{cases} S_3(S_1(x_R) \oplus x_L) \oplus S_3(S_1(x_R \oplus a) \oplus x_L \oplus b) = 0, \\ S_2(x_L) \oplus S_1(x_R) \oplus x_L \oplus S_2(x_L \oplus b) \oplus S_1(x_R \oplus a) \oplus x_L \oplus b = c \end{cases}$$
$$\Leftrightarrow \begin{cases} S_3(S_1(x_R) \oplus x_L) \oplus S_3(S_1(x_R \oplus a) \oplus x_L \oplus b) = 0, \\ S_2(x_L) \oplus S_1(x_R) \oplus S_2(x_L \oplus b) \oplus S_1(x_R \oplus a) = b \oplus c \end{cases} .$$

Cela signifie de manière équivalente qu'il existe un $d \in \mathbb{F}_2^n$ tel que

$$\begin{cases} x_R \in D_{S_1}(a \to b \oplus d),\ x_L \in D_{S_2}(b \to c \oplus d), \\ S_3(S_1(x_R) \oplus x_L) \oplus S_3(S_1(x_R \oplus a) \oplus x_L \oplus b) = 0 , \end{cases}$$

i.e.,

$$x_R \in D_{S_1}(a \to b \oplus d),\ x_L \in D_{S_2}(b \to c \oplus d) \text{ et } S_1(x_R) \oplus x_L \in D_{S_3}(d \to 0) .$$

Alors, pour tout $d \in \mathbb{F}_2^n$ fixé tel que $\delta_{S_3}(d, 0) = 0$, aucune paire (x_L, x_R) ne satisfait la troisième condition. Si $\delta_{S_3}(d, 0) > 0$, alors certaines des valeurs (x_L, x_R) définies par les deux premières conditions peuvent aussi vérifier la troisième, et si $d = 0$, la troisième solution est toujours vérifiée. Il s'ensuit que

$$\delta_{S_1}(a,b) \times \delta_{S_2}(b,c) \leqslant \delta_F(b\|a, c\|c) \leqslant \sum_{d \in \mathbb{F}_2^n} \delta_{S_1}(a, b \oplus d) \times \delta_{S_2}(b, c \oplus d) \times \gamma_{S_3}(d)$$

où $\gamma_{S_3}(d)$ vaut 0 si $\delta_{S_3}(d, 0) = 0$ et 1 sinon. Qui plus est, si S_3 est bijective, $\delta_{S_3}(d, 0) > 0$ si et seulement si $d = 0$, ce qui implique que les deux bornes précédentes sont égales, i.e.,

$$\delta_F(b\|a, c\|c) = \delta_{S_1}(a,b) \times \delta_{S_2}(b,c) . \qquad \square$$

Borne générale.

Ces trois différentielles particulières nous procurent la borne inférieure suivante sur l'uniformité différentielle de 3 tours de réseau de type MISTY.

Théorème 2.13. *Soit F la fonction obtenue par 3 tours d'un réseau de type MISTY utilisant pour fonctions de tour les boîtes-S de n bits S_1, S_2 et S_3. Alors*

$$\delta(F) \geqslant \delta(S_1) \max\left(\delta_{\min}(S_2), \delta_{\min}(S_3)\right) ,$$

où $\delta_{\min}(S) = \min_{a \neq 0} \max_b \delta_S(a,b)$. De plus,

— *si S_1 est une permutation,*

$$\delta(F) \geqslant \max_{i\neq 1, j\neq 1} \left(\delta(S_i)\delta_{\min}(S_j),\ \delta(S_i)\delta_{\min}(S_1^{-1})\right) ,$$

— *si S_1 n'est pas une permutation, $\delta(F) \geqslant 2^{n+1}$.*

Démonstration. Ce résultat est une conséquence directe du théorème 2.12. Nous dérivons ici les bornes du premier item du théorème 2.12, les autres cas peuvent être déduits de façon similaire des deux autres items.

Considérons tout d'abord une paire de différences (α, β) qui atteint l'uniformité différentielle de S_1, i.e., $\delta(S_1) = \delta_{S_1}(\alpha, \beta)$.

Puis, choisissons $a = \alpha$ et $c = \beta$, nous obtenons que, pour tout $b \in \mathbb{F}_2^n$,

$$\delta_F(0\|\alpha, b\|\beta) = \delta(S_1) \times \delta_{S_3}(\beta, \beta \oplus b) .$$

Ensuite, nous pouvons choisir pour b la valeur qui maximise $\delta_{S_3}(\beta, \beta \oplus b)$. Cette valeur est toujours supérieure ou égale à $\delta_{\min}(S_3)$.

De même, considérons désormais une paire de différences (α, β) qui atteint l'uniformité différentielle de S_3, i.e., $\delta(S_3) = \delta_{S_3}(\alpha, \beta)$.

Dans ce cas, nous choisissons $c = \alpha$ et $b = \alpha \oplus \beta$, et obtenons que, pour tout $a \in \mathbb{F}_2^n$,

$$\delta_F(0\|a, (\alpha \oplus \beta)\|\alpha) = \delta_{S_1}(a, \alpha) \times \delta(S_3) .$$

Nous choisissons alors pour a la valeur qui maximise $\delta_{S_1}(a, \alpha)$ et qui est toujours supérieure ou égale à $\delta_{\min}(S_1^{-1})$ lorsque S_1 est une permutation.

Supposons maintenant que S_1 n'est pas bijective. Cela signifie qu'il existe un $a \in \mathbb{F}_2^n$ non-nul tel que $\delta_{S_1}(a, 0) \geqslant 2$. Alors nous déduisons du premier item du théorème 2.12, avec $b = c = 0$, que l'équation

$$F(x_L\|x_R) \oplus F_K(x_L \oplus a\|x_R) = (0, 0)$$

a

$$\delta_{S_1}(a, 0) \times \delta_{S_3}(0, 0) \geqslant 2 \times 2^n = 2^{n+1}$$

solutions dans $\mathbb{F}_2^{2n}$. □

Application à $n = 4$.

C'est cette application aux réseaux de type MISTY sur 2×4 bits qui diffère du cas des réseaux de Feistel. Tous les résultats sont similaires au cas des réseaux de Feistel, toutefois les conditions nécessaires pour obtenir une permutation de 8 bits sont incompatibles avec les conditions nécessaires pour obtenir une fonction de 8 bits optimalement résistante.

Proposition 2.4. *Soit F la fonction obtenue par 3 tours d'un réseau de type MISTY utilisant pour fonctions de tour les boîtes-S de 4 bits S_1, S_2 et S_3. Alors*

$$\delta(F) \geqslant 8 .$$

Démonstration.
Il s'agit d'une conséquence du théorème 2.12.

Si S_1 n'est pas une permutation, alors la borne générale donne $\delta(F) \geqslant 2^5 = 32$.

Si S_1 est une permutation, on sait qu'il n'existe pas de permutation APN sur 4 bits, donc $\delta(S_1) \geqslant 4$, et par conséquent $\delta(F) \geqslant \delta(S_1)\delta_{\min}(S_2) \geqslant 4 \times 2 = 8$. □

Là encore, nous déduisons des conditions nécessaires pour que F atteigne $\delta(F) = 8$.

Comme dans le cas des réseaux de Feistel, la partie la plus compliquée consiste à observer des propriétés sur les boîtes-S de 4 bits. Les propriétés sont les mêmes que dans le cas des réseaux de Feistel, à savoir les Propriétés 2.3 et 2.4.

Lemme 2.5. *Soit F la fonction obtenue par 3 tours d'un réseau de type MISTY utilisant pour fonctions de tour les boîtes-S de 4 bits S_1, S_2 et S_3. Supposons que S_1 est une permutation avec $\delta(S_1) = 4$. Dans ce cas, si $\delta(S_2) \geqslant 4$ ou $\delta(S_3) \geqslant 4$, alors $\delta(F) \geqslant 16$.*

Démonstration.
L'idée est exactement la même que pour les réseaux de Feistel.

Supposons que S_3 est telle que $\delta(S_3) \geqslant 4$. La Propriété 2.3 prouve qu'il existe au moins 3 lignes a, c et $a \oplus c$ non-nulles de la table des différences de S_3 qui contiennent au moins une valeur supérieure ou égale à 4. La Propriété 2.4 prouve que, parmi ces 3 différences a, c et $a \oplus c$, au moins une correspond à une colonne de S_2 qui contient un 4.

Par le premier item du théorème 2.12, on obtient une différentielle avec au moins 16 solutions. □

Théorème 2.14. *Soit F la fonction obtenue par 3 tours d'un réseau de type MISTY utilisant pour fonctions de tour les boîtes-S de 4 bits S_1, S_2 et S_3. Alors les conditions suivantes sont nécessaires pour avoir $\delta(F) = 8$:*

- *S_1 une permutation avec $\delta(S_1) = 4$,*
- *S_2 APN (donc pas une permutation),*
- *S_3 APN (donc pas une permutation).*

Sinon, $\delta(F) \geqslant 12$.

Démonstration.
D'après le théorème 2.12, il faut que S_1 soit une permutation, sinon $\delta(F) \geqslant 32$. Par ailleurs, si $\delta(S_1) \geqslant 6$, alors $\delta(F) \geqslant 12$, donc il faut $\delta(S_1) \leqslant 4$. Or il n'existe pas de permutation de 4 bits APN, donc il faut $\delta(S_1) = 4$.

D'après le lemme 2.5, si S_2 ou S_3 n'est pas APN, alors $\delta(F) \geqslant 16$.

D'où le résultat. □

Rappelons que les boîtes-S les plus intéressantes sont les boîtes-S bijectives (permutations), et qu'un réseau de type MISTY n'est bijectif que si S_1, S_2 et S_3 sont bijectives. On constate que les conditions du théorème 2.14 ne peuvent donc pas être vérifiées pour obtenir une boîte-S bijective (puisqu'il n'existe pas S_2 et S_3 bijectives et APN sur 4 bits). D'après le théorème 2.14, on déduit donc que toute permutation F obtenue par 3 tours de réseau de type MISTY vérifie $\delta(F) \geqslant 12$.

En observant des propriétés détaillées des permutations de 4 bits, on peut même prouver que toute permutation F obtenue par 3 tours de réseau de type MISTY vérifie $\delta(F) \geqslant 16$. Les propriétés des permutations de 4 bits qui nous intéressent sont les suivantes :

Lemme 2.6. *Soient S_1, S_2 et S_3 des permutations de 4 bits. Alors il existe une différence non-nulle $\gamma \in \mathbb{F}_2^4 \backslash \{0\}$ telle qu'au moins l'une de ces propositions est vérifiée :*

- *La table des différences de S_1 contient au moins une valeur supérieure ou égale à 4 en colonne γ et la table des différences de S_2 contient au moins une valeur supérieure ou égale à 4 à la ligne γ ;*
- *La table des différences de S_1 contient au moins une valeur supérieure ou égale à 4 en colonne γ et la table des différences de S_3 contient au moins une valeur supérieure ou égale à 4 à la ligne γ ;*
- *La table des différences de S_2 contient au moins une valeur supérieure ou égale à 4 à la ligne γ et la table des différences de S_3 contient au moins une valeur supérieure ou égale à 4 à la ligne γ ;*

Démonstration. Nous obtenons ce résultat par une recherche exhaustive sur les classes d'équivalence affine, exactement comme pour la classification des boîtes-S de 4 bits optimales dans [De 07 ; LP07b]. Il y a 302 classes d'équivalence affine pour les permutations de 4 bits. Nous avons sélectionné un représentant de chaque classe d'équivalence affine, puis nous avons vérifié que leurs tables des différences contiennent au moins six lignes définies

par une certaine différence en entrée non-nulle a qui contiennent une valeur supérieure ou égale à 4. Notons $\mathcal{R}(S)$ l'ensemble correspondant (de taille au moins 6) :

$$\mathcal{R}(S) = \left\{a \in \mathbb{F}_2^4 \backslash \{0\} \ : \exists b \in \mathbb{F}_2^4 \backslash \{0\}, \delta_S(a,b) \geqslant 4\right\} .$$

De ce fait, s'il n'existe aucune différence $\gamma \in \mathbb{F}_2^4 \backslash \{0\}$ qui satisfasse l'une des trois propositions du lemme, alors on aurait que les trois ensembles $\mathcal{R}(S_2)$, $\mathcal{R}(S_3)$ et $\mathcal{R}(S_1^{-1})$ sont disjoints. En d'autres termes, nous pourrions trouver 18 valeurs distinctes parmi les 15 éléments non-nuls de $\mathbb{F}_2^4$, ce qui est impossible. □

Corollaire 2.1 (MISTY bijectif sur 8 bits)**.** *Soit F une permutation obtenue par 3 tours d'un réseau de type MISTY utilisant pour fonctions de tour les boîtes-S de 4 bits S_1, S_2 et S_3. Alors*

$$\delta(F) \geqslant 16 .$$

Démonstration.
Ce résultat dérive directement du théorème 2.12 combiné au lemme 2.6. En effet, le lemme 2.6 garantit l'existence de a, b et c tels qu'au moins l'une de ces trois propriétés est vérifiée :

— $\delta_{S_1}(a,c) \geqslant 4$ et $\delta_{S_3}(c, b \oplus c) \geqslant 4$,
— $\delta_{S_2}(a, a \oplus c) \geqslant 4$ et $\delta_{S_3}(a, b \oplus c) \geqslant 4$,
— $\delta_{S_1}(a,b) \geqslant 4$ et $\delta_{S_2}(b,c) \geqslant 4$.

Dans chacune de ces trois situations, le théorème 2.12 exhibe une différentielle (α, β) à travers F telle que $\delta_F(\alpha, \beta) = 16$. □

Ainsi, même si les bornes et les conditions sont similaires entre les réseaux de type MISTY et les réseaux de Feistel, on observe ici que la restriction imposée par le réseau de type MISTY lorsqu'on veut une permutation nous empêche d'obtenir des résultats optimaux en termes d'uniformité différentielle.

Pour cette raison, je préconise davantage d'utiliser des réseaux de Feistel, et c'est ce que nous ferons lors des instanciations (cf. section 2.4.1).

2.2.3.4 Linéarité de 3 tours de MISTY

La borne inférieure sur la linéarité de 3 tours d'un réseau de type MISTY peut être obtenue de façon similaire à l'uniformité différentielle (et similaire au cas des réseaux de Feistel).

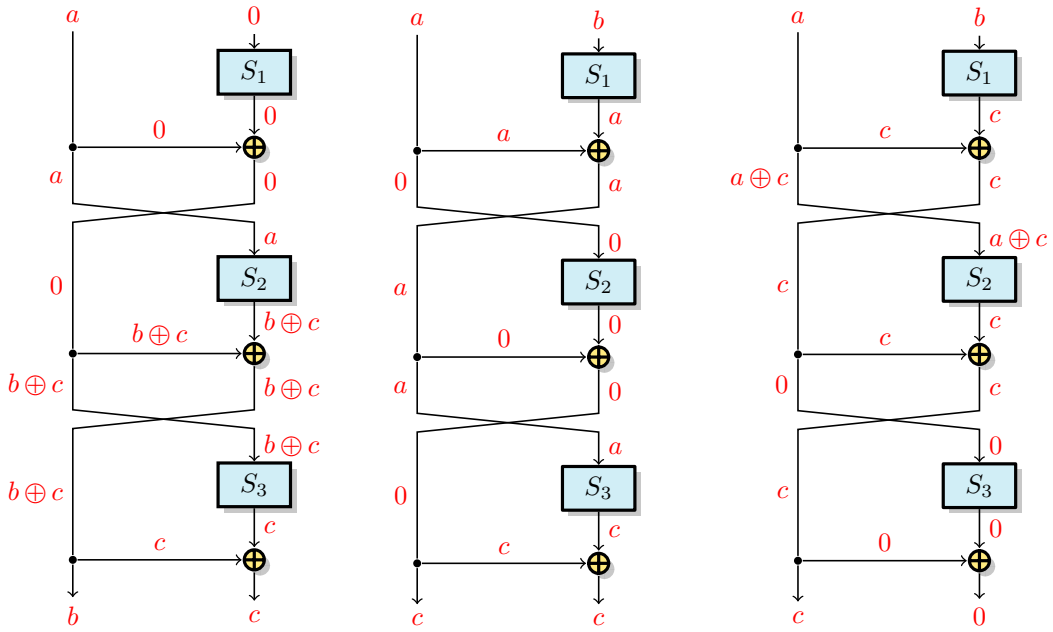
Les masques annulant une boîte-S.

Théorème 2.15. *Soit F une permutation obtenue par 3 tours d'un réseau de type MISTY utilisant pour fonctions de tour les boîtes-S de 4 bits S_1, S_2 et S_3. Alors, pour tous a, b et c dans $\mathbb{F}_2^n$, on a :*

(i) $\hat{F}(a\|b, 0\|c) = \hat{S}_1(b,c)\hat{S}_2(a \oplus c, c)$
(ii) $\hat{F}(a\|b, c\|c) = \hat{S}_1(b,a)\hat{S}_3(a,c)$
(iii) *Si S_1 est bijective,* $\hat{F}(a\|0, b\|c) = \hat{S}_2(a, b \oplus c)\hat{S}_3(b \oplus c, b)$

Démonstration. Les trois résultats correspondent aux configurations illustrées en figures 2.9a, 2.9b et 2.9c.

(i)
$$\begin{aligned}\hat{F}(a\|b, 0\|c) &= \sum_{(x_R, x_L) \in (\mathbb{F}_2^n)^2} (-1)^{c \cdot S_2(x_L) \oplus c \cdot S_1(x_R) \oplus c \cdot x_L \oplus a \cdot x_L \oplus b \cdot x_R} \\ &= \sum_{x_R \in \mathbb{F}_2^n} (-1)^{c \cdot S_1(x_R) \oplus b \cdot x_R} \sum_{x_L \in \mathbb{F}_2^n} (-1)^{c \cdot S_2(x_L) \oplus (a \oplus c) \cdot x_L} \\ &= \hat{S}_1(b,c)\, \hat{S}_2(a \oplus c, c)\end{aligned}$$



(a) Masques $(a||0, b||c)$ sur 3 tours d'un réseau de MISTY qui annulent le masque en entrée de S_1.

(b) Masques $(a||b, c||c)$ sur 3 tours d'un réseau de MISTY qui annulent le masque en entrée de S_2.

(c) Masques $(a||b, c||0)$ sur 3 tours d'un réseau de MISTY qui annulent le masque en entrée de S_3.

Figure 2.9 – Propagation des masques linéaires dans 3 tours d'un réseau de type MISTY pour les masques annulant les fonctions internes. Les valeurs en rouge sont des masques linéaires.

(ii)
$$\begin{aligned}\hat{F}(a\|b,c\|c) &= \sum_{(x_R,x_L)\in(\mathbb{F}_2^n)^2}(-1)^{(c,c)\cdot F_K(x_L\|x_R)\oplus(a,b)\cdot(x_L,x_R)}\\ &= \sum_{(x_R,x_L)\in(\mathbb{F}_2^n)^2}(-1)^{c\cdot S_3(S_1(x_R)\oplus x_L)\oplus a\cdot x_L\oplus b\cdot x_R}\end{aligned}$$

Nous fixons $x_L = S_1(x_R)\oplus z$ et observons que, pour tout x_R fixé, z prend toutes les valeurs possibles dans $\mathbb{F}_2^n$ lorsque x_L varie, ce qui implique que

$$\hat{F}(a\|b,c\|c) = \sum_{x_R\in\mathbb{F}_2^n}\sum_{z\in\mathbb{F}_2^n}(-1)^{c\cdot S_3(z)\oplus a\cdot(z\oplus S_1(x_R))\oplus b\cdot x_R} = \hat{S}_1(b,a)\hat{S}_3(a,c)\ .$$

(iii)
$$\begin{aligned}\hat{F}(a\|0,b\|c) &= \sum_{(x_R,x_L)\in(\mathbb{F}_2^n)^2}(-1)^{(b,c)\cdot F_K(x_L\|x_R)\oplus(a,0)\cdot(x_L,x_R)}\\ &= \sum_{(x_R,x_L)\in(\mathbb{F}_2^n)^2}(-1)^{b\cdot S_3(S_1(x_R)\oplus x_L)\oplus[b\oplus c]\cdot[S_1(x_R)\oplus S_2(x_L)]\oplus[a\oplus b\oplus c]\cdot x_L}\end{aligned}$$

Si S_1 est bijective, nous fixons $x_R = S_1^{-1}(z\oplus x_L)$. En utilisant que, pour tout x_L fixé, z prend toutes les valeurs possibles dans $\mathbb{F}_2^n$ lorsque x_R varie, nous déduisons que

$$\begin{aligned}\hat{F}(a\|0,b\|c) &= \sum_{x_L\in\mathbb{F}_2^n}(-1)^{(b\oplus c)\cdot S_2(x_L)\oplus a\cdot x_L}\sum_{z\in\mathbb{F}_2^n}(-1)^{b\cdot S_3(z)\oplus(b\oplus c)\cdot z}\\ &= \hat{S}_2(a,b\oplus c)\hat{S}_3(b\oplus c,b)\end{aligned}$$

□

Borne générale.

Comme pour le cas différentiel, les trois approximations linéaires précédentes nous procurent une borne inférieure sur la linéarité.

Théorème 2.16. *Soit F une permutation obtenue par 3 tours d'un réseau de type MISTY utilisant pour fonctions de tour les boîtes-S de 4 bits S_1, S_2 et S_3. Alors*

$$\mathcal{L}(F) \geqslant \max\left(\mathcal{L}(S_1)\mathcal{L}_{\min}(S_2),\ \mathcal{L}(S_2)\mathcal{L}_{\min}(S_1),\ \mathcal{L}(S_3)\mathcal{L}_{\min}(S_1)\right) .$$

De plus
- *si S_1 est une permutation, $\mathcal{L}(F) \geqslant \mathcal{L}(S_3)\mathcal{L}_{\min}(S_2)$,*
- *si S_3 est une permutation, $\mathcal{L}(F) \geqslant \mathcal{L}(S_1)\mathcal{L}_{\min}(S_3^{-1})$,*
- *si S_1 et S_3 sont des permutations, alors $\mathcal{L}(F) \geqslant \mathcal{L}(S_2)\mathcal{L}_{\min}(S_3^{-1})$.*

Démonstration. Choisissons tout d'abord une paire de masques (α, β) pour une S_i qui atteint la linéarité $\mathcal{L}(S_i)$. Pour $i = 1$ ou $i = 2$, nous utilisons le premier item du théorème 2.15 et déduisons que, pour tout γ,

$$\begin{aligned} |\hat{F}(\gamma\|\alpha, 0\|\beta)| &= \mathcal{L}(S_1)|\hat{S_2}(\gamma \oplus \beta, \beta)| \\ |\hat{F}((\alpha \oplus \beta)\|\gamma, 0\|\beta)| &= \mathcal{L}(S_2)|\hat{S_1}(\gamma, \beta)| . \end{aligned}$$

Pour $i = 3$, nous utilisons le deuxième item du théorème 2.15 et obtenons

$$|\hat{F}(\alpha\|\gamma, \beta\|\beta)| = \mathcal{L}(S_3)|\hat{S_1}(\gamma, \alpha)| .$$

De plus, lorsque S_1 est une permutation, le troisième item du théorème 2.15 s'applique, et pour (α, β) tels que $|\hat{S_3}(\alpha, \beta)| = \mathcal{L}(S_3)$, nous obtenons

$$|\hat{F}(\gamma\|0, \beta\|(\alpha \oplus \beta))| = \mathcal{L}(S_3)|\hat{S_2}(\gamma, \alpha)| .$$

Ensuite, dans chacun de ces quatre cas, nous choisissons pour γ la valeur non-nulle qui maximise le terme de droite dans le produit, i.e., qui maximise la transformée de Walsh de la composante impliquée de S_j. Par définition, $\mathcal{L}_{\min}(S_j)$ est alors une borne inférieure pour le terme de droite.

Les dernières affirmations dans le théorème sont dérivées du deuxième (resp. troisième) item du théorème 2.15, en choisissant (α, β) tels que $|\hat{S_1}(\alpha, \beta)| = \mathcal{L}(S_1)$ (resp. $|\hat{S_2}(\alpha, \beta)| = \mathcal{L}(S_2)$). Alors non obtenons

$$\begin{aligned} |\hat{F}(\beta\|\alpha, \gamma\|\gamma)| &= \mathcal{L}(S_1)|\hat{S_3}(\beta, \gamma)| \\ |\hat{F}(\alpha\|0, \gamma\|\beta \oplus \gamma)| &= \mathcal{L}(S_2)|\hat{S_3}(\beta, \gamma)| , \end{aligned}$$

où la seconde inégalité tient lorsque S_1 est une permutation. Puis, si S_3 est une permutation, nous utilisons que, pour tout $\beta \neq 0$ fixé,

$$\max_{\gamma \in \mathbb{F}_2^n} |\hat{S_3}(\beta, \gamma)| \geqslant \mathcal{L}_{\min}(S_3^{-1}) . \qquad \square$$

Application à $n = 4$

Pour ce qui est de la linéarité, le réseaux de type MISTY de 8 bits sur 3 tours se comporte exactement comme le réseau de Feistel sur 3 tours. En effet, la linéarité des permutations de 4 bits n'est pas moins bonne que celle des fonctions non-bijectives.

Il est connu que la meilleure linéarité atteignable pour une permutation de 4 bits est 8 (en particulier, c'est un corollaire du lemme 2.2), et nous montrons maintenant que cette propriété est aussi vérifiée pour les fonctions non-bijectives.

Lemme 2.7. *Toute boîte-S de 4 bits satisfait* $\mathcal{L}(S) \geqslant 8$.

Démonstration. Supposons qu'il existe S de $\mathbb{F}_2^4$ dans $\mathbb{F}_2^4$ avec $\mathcal{L}(S) < 8$, *i.e.* avec $\mathcal{L}(S) \leqslant 6$. Alors, toutes les composantes non-nulles de S, $S_c : x \mapsto c \cdot S(x)$ avec $c \neq 0$, satisfont $\mathcal{L}(S_c) \leqslant 6$. D'après la classification des fonctions booléennes d'au plus 5 variables par Berlekamp et Welch [BW72], nous déduisons que tout S_c, $c \neq 0$, est équivalent affine soit à $x_1x_2x_3x_4 + x_1x_2 + x_3x_4$ soit à $x_1x_2 + x_3x_4$, car ce sont les seules classes de fonctions booléennes de linéarité au plus 6.

Notons L_1 (resp. L_2) l'ensemble des c non-nuls, $c \in \mathbb{F}_2^4$, tels que S_c appartienne à la première (resp. seconde) classe. Puisque le degré est invariant par transformations affines, L_1 (resp. L_2) correspondent aux composantes de degré 4 (resp. de degré au plus 2). La somme de deux composantes de degré au plus 2 est de degré au plus 2, impliquant que $L_2 \cup \{0\}$ est un espace vectoriel V de $\mathbb{F}_2^4$. Il s'ensuit que la projection de S sur V peut être vue comme une fonction de $\mathbb{F}_2^4$ dans $\mathbb{F}_2^{\dim V}$ de linéarité 4, i.e., une fonction courbe.

Dans [Nyb91b], Nyberg montre que, si une fonction F de $\mathbb{F}_2^n$ dans $\mathbb{F}_2^m$ est courbe, alors $m \leqslant n/2$. Ainsi, $\dim V \leqslant 2$. Mais la somme de deux composantes quelconques S_c de degré 4 ne peut pas avoir degré 4 puisqu'il existe un seul monôme de degré 4 à 4 variables. Ainsi, les $\binom{t}{2}$ sommes de deux composantes de L_1 sont dans L_2. De plus, si L_1 contient t mots de poids 1 (*i.e.*, si S a t coordonnées de linéarité 6), alors L_2 contient $4 - t$ mots de poids 1 dont les $2^{4-t} - 1$ combinaisons linéaires sont dans L_2. Ainsi,

$$\#L_2 \geqslant \binom{t}{2} + 2^{4-t} - 1 > 3,$$

pour tout $0 \leqslant t \leqslant 4$, ce qui est contradictoire. □

Combiné au lemme 2.7, le théorème 2.16 procure la borne inférieure suivante sur la linéarité de 3 tours d'un réseau de type MISTY sur $\mathbb{F}_2^8$.

Corollaire 2.2. *Toute fonction de 8 bits F correspondant à 3 tours de réseau de type MISTY satisfait* $\mathcal{L}(F) \geqslant 32$.

Cette borne n'a qu'un intérêt marginal puisque, à notre connaissance, $\mathcal{L}(S) = 32$ est la meilleure linéarité connue pour une boîte-S de 8 bits. Mais une fois de plus, cette borne inférieure peut être améliorée lorsqu'on se concentre sur les permutations. En effet, nous pouvons exploiter que $\mathcal{L}_{\min}(S) \geqslant 8$ pour toute permutation de 4 bits (lemme 2.2).

Du fait que toute permutation de 4 bits S satisfait $\mathcal{L}(S) \geqslant 8$ et $\mathcal{L}_{\min}(S) \geqslant 8$, nous déduisons directement du théorème 2.16 la borne inférieure plus précise suivante :

Proposition 2.5. *Soit F une permutation obtenue par 3 tours d'un réseau de type MISTY utilisant pour fonctions de tour les boîtes-S de 4 bits S_1, S_2 et S_3. Si n'importe quelle boîte-S parmi S_1, S_2 et S_3 est une permutation, alors $\mathcal{L}(F) \geqslant 64$. Notablement, si $\mathcal{L}(F) < 64$, alors $\delta(F) \geqslant 32$.*

La dernière affirmation de la proposition est déduite du premier item du théorème 2.13. Bien que ce résultat montre que 3 tours d'un réseau de type MISTY avec $\mathcal{L}(F) < 64$ serait de peu d'intérêt, nous montrons maintenant que leur linéarité est au moins de 48 (similairement au cas des réseaux de Feistel).

Nous réutilisons ici les résultats démontrés au paragraphe 2.2.3.2.

Théorème 2.17. *Soit F une permutation obtenue par 3 tours d'un réseau de type MISTY utilisant pour fonctions de tour les boîtes-S de 4 bits S_1, S_2 et S_3. Alors $\mathcal{L}(F) \geqslant 48$.*

Démonstration. Le résultat est immédiat s'il existe c non-nul avec $|\hat{S}_3(0, c)| \geqslant 4$. Sinon, nous choisissons a et b tels que $|\hat{S}_1(b, a)| \geqslant 8$, et les lemmes 2.3 et 2.4 montrent l'existence d'un c avec $|\hat{S}_3(a, c)| \geqslant 6$. Nous obtenons que $|\hat{F}(a\|b, c\|c)| \geqslant 48$ par le théorème 2.15. □

Nous conjecturons que tout réseau de type MISTY de 8 bits sur 3 tours vérifie même que $\mathcal{L}(F) \geqslant 64$, mais là encore ce résultat semble difficile à prouver sans classification plus précise des fonctions de 4 bits.

2.3 Boîtes-S à bas coût, une étude zoologique : Papillons

Les paragraphes suivants détaillent les études et les résultats obtenus sur les schémas de Papillons et leur généralisation. Une comparaison avec les réseaux de Feistel et de type MISTY est donnée par la suite en section 2.4.3.

La généralisation des Papillons est un travail publié à IEEE-IT [CDP17]. Pour les définitions, se référer à la section 1.3.

2.3.1 Travaux antérieurs : les Papillons

Dans leur article de Crypto 2016 [PUB16], Perrin *et al.* introduisent le Papillon et entament une étude de la sécurité de celui-ci.

Tout d'abord, énonçons clairement la définition d'un Papillon :

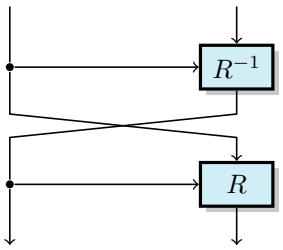


Figure 2.10(a): Papillon ouvert (représentation avec R).

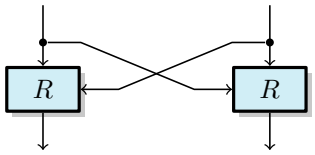


Figure 2.10(b): Papillon fermé (représentation avec R).

Définition 2.4 (Papillons)**.** Soit $\alpha \in \mathbb{F}_{2^n}$, e un entier tel que $x \mapsto x^e$ soit une permutation de $\mathbb{F}_{2^n}$ et $R_k[e, \alpha]$ la permutation à clef :

$$R_k[e, \alpha](x) = (x + \alpha k)^e + k^e \,.$$

On appelle Papillons les fonctions de $(\mathbb{F}_{2^n})^2$ dans $(\mathbb{F}_{2^n})^2$ définies par :

— on appelle Papillon ouvert de taille de branche n, d'exposant e et de coefficient α la permutation notée H^e_α définie par :

$$\mathsf{H}^e_\alpha(x, y) = \left(R^{-1}_{R_y[e,\alpha](x)}(y),\, R_y[e\alpha](x)\right),$$

— on appelle Papillon fermé de taille de branche n, d'exposant e et de coefficient α la fonction notée V^e_α définie par :

$$\mathsf{V}^e_\alpha(x, y) = (R_y[e, \alpha](x),\, R_x[e, \alpha](y)) \,.$$

De plus, ces deux structures sont équivalentes CCZ.

Le cas le plus intéressant est celui où le degré algébrique de R est 2, c'est-à-dire lorsque x^e est de degré algébrique 2, en particulier e de la forme 3×2^i. En effet, dans ce cas, le Papillon fermé est de degré algébrique 2 (quadratique), ce qui est bien plus simple à analyser qu'un cas général. En particulier, sa dérivée $x \mapsto F(x) \oplus F(x \oplus dx)$ est linéaire,

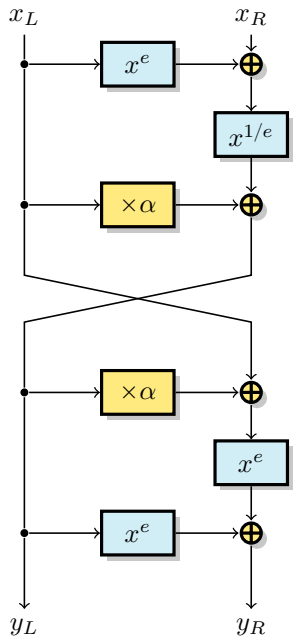


Figure 2.11(a): Papillon ouvert (décomposition de R).

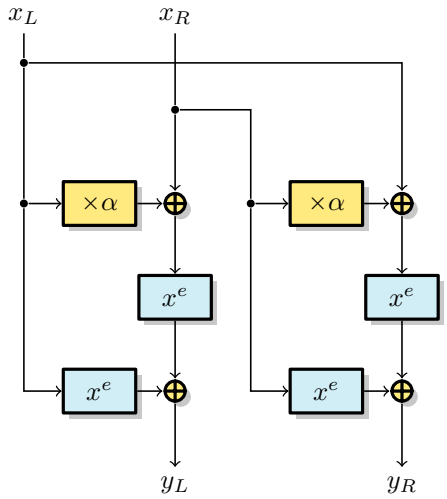


Figure 2.11(b): Papillon fermé (décomposition de R).

et c'est cette dérivée qui nous intéresse dans le calcul de l'uniformité différentielle (et de la linéarité indirectement).

Une fois calculée la sécurité du Papillon fermé, l'équivalence CCZ entre le Papillon fermé et le Papillon ouvert nous donne directement que la sécurité du Papillon ouvert (quant aux attaques différentielles et linéaires) est la même que pour le Papillon fermé.

Dans ce cas où le Papillon fermé est quadratique, Perrin *et al.* ont pu utiliser des résultats bien connus sur les fonctions booléennes pour établir la sécurité du Papillon.

Les résultats sur la sécurité des Papillons obtenus par Perrin *et al.* sont les suivants :

Théorème 2.18 (Sécurité d'un Papillon (Perrin *et al.* [PUB16])**.** *Soient* V^e_α *et* H^e_α *respectivement les Papillons fermé et ouvert de* $2n$ *bits d'exposant* $e = 3 \times 2^t$ *pour un certain* t*, de coefficient* $\alpha \notin \{0, 1\}$ *et* n *impair. Alors :*

$$\delta(\mathsf{V}^e_\alpha) \leqslant 4\,,$$

$$\delta(\mathsf{H}^e_\alpha) \leqslant 4\,.$$

De plus, V^e_α *est quadratique et la moitié des coordonnées de* H^e_α *sont de degré algébrique* n*, l'autre moitié de degré* $n + 1$*.*

De plus, un résultat essentiel est que, sur 6 bits, la boîte-S de Dillon est équivalente affine à un Papillon ouvert. En particulier, sur 6 bits, il existe des Papillons APN.

À l'évidence, il manque à cette étude la résistance aux attaques linéaires ($\mathcal{L}$), et une question primordiale se pose : existe-t-il d'autres Papillons APN que celui trouvé sur 6 bits et qui est équivalent à la boîte-S de Dillon ?

Dans un article publié à IEEE-IT en 2017, Anne Canteaut, Léo Perrin et moi-même avons répondu à ces questions ouvertes de [PUB16] en commençant par généraliser les Papillons.

2.3.2 Généralisation des Papillons

Soit $m = 2n$ un entier pair. Dans tous ces travaux, les fonctions booléennes à m variables sont identifiées aux fonctions sur $\mathbb{F}_{2^n} \times \mathbb{F}_{2^n}$. Similairement, les fonctions vectorielles de $\mathbb{F}_2^m$ dans $\mathbb{F}_2^m$ sont identifiées aux applications de $\mathbb{F}_{2^n} \times \mathbb{F}_{2^n}$ dans lui-même. Notons que le choix de la base utilisée pour identifier $\mathbb{F}_{2^n}$ à $\mathbb{F}_2^n$ n'affecte pas les propriétés cryptographiques des fonctions que nous étudions puisque des bases différentes mènent à des fonctions qui sont équivalentes affines.

Dans ce contexte, le produit scalaire entre deux éléments (x_1, y_1) et (x_2, y_2) dans $\mathbb{F}_{2^n} \times \mathbb{F}_{2^n}$ est défini par

$$\mathrm{Tr}(x_1 x_2) + \mathrm{Tr}(y_1 y_2)$$

où Tr est la fonction trace sur $\mathbb{F}_{2^n}$, i.e., $\mathrm{Tr}(x) = x + x^2 + \ldots + x^{2^{n-1}}$.

Définissons maintenant la famille de fonctions vectorielles que nous étudierons dans ces travaux.

Définition 2.5 (Papillons généralisés)**.** Soit R un polynôme bivarié de $\mathbb{F}_{2^n}$ tel que $R_y : x \mapsto R(x, y)$ soit une permutation de $\mathbb{F}_{2^n}$ pour tout y dans $\mathbb{F}_{2^n}$. Le *Papillon fermé* V_R est la fonction de $(\mathbb{F}_{2^n})^2$ définie par

$$\mathsf{V}_R(x, y) \;=\; \big(R(x, y),\; R(y, x)\big)$$

et le *Papillon ouvert* H_R est la permutation de $(\mathbb{F}_{2^n})^2$ définie par

$$\mathsf{H}_R(x, y) \;=\; \big(R_{R_y^{-1}(x)}(y),\; R_y^{-1}(x)\big)$$

où $R_y(x) = R(x, y)$ et $R_y^{-1}\left(R_y(x)\right) = x$ pour tous y, x. Une représentation de H_R est donnée en figure 2.12a et une de V_R en figure 2.12b.

Il est aisé de vérifier que, pour tout choix de permutation à clef R, le Papillon ouvert H_R est une involution.

Dans ces travaux, nous nous restreignons au cas où $R(x, y)$ est de *degré univarié 3* tout comme les Papillons de [PUB16]. Ainsi, l'uniformité différentielle et la linéarité des Papillons généralisés seront uniquement calculés pour le Papillon fermé correspondantV_R qui est seulement de degré algébrique 2.

Le lemme suivant décrit tous les polynômes R satisfaisant cette condition de degré qui définissent une permutation à clef, tel que le demande la définition des Papillons. D'un point de vue cryptographique, le fait que R corresponde à une permutation à clef peut être vu comme une propriété intégrale, comme il est remarqué dans [PUB16].

Lemme 2.8 (Restriction sur le degré)**.** *Soit R un polynôme bivarié de $\mathbb{F}_{2^n}$ tel que $R_y : x \mapsto R(x, y)$ soit une permutation pour tout y et tel que tous les termes dans R soient non-linéaires de degré au plus 3. Alors R peut être décrit en utilisant deux éléments de $\mathbb{F}_{2^n}$ notés α et β par*

$$R(x, y) = (x + \alpha y)^3 + \beta y^3 \,.$$

Nous notons les Papillons construits sur de tels polynômes R par $\mathsf{V}_{\alpha,\beta}$ et $\mathsf{H}_{\alpha,\beta}$ pour les Papillons fermés et ouverts respectivement.

La preuve de ce lemme repose sur le théorème suivant.

Théorème 2.19 (Corollaire 2.9 de [MS87])**.** *Soit $\mathbb{F}_q$ de caractéristique autre que 3. Alors $f : x \mapsto ax^3 + bx^2 + cx + d$ $(a \neq 0)$ permute $\mathbb{F}_q$ si et seulement si $b^2 = 3ac$ et $q \equiv 2 \mod 3$.*

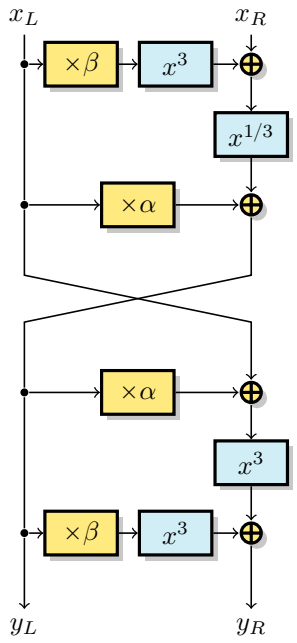


Figure 2.12(a): Papillon ouvert généralisé. On étudie l'exposant 3, mais le résultat se généralise pour tout exposant de la forme 3×2^i.

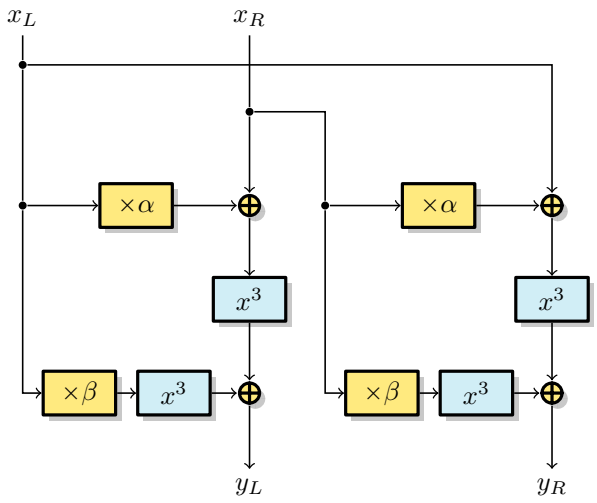


Figure 2.12(b): Papillon fermé généralisé. On étudie l'exposant 3, mais le résultat se généralise pour tout exposant de la forme 3×2^i.

Preuve du lemme 2.10. Soit

$$R(x, y) = Ax^3 + Bx^2y + Cxy^2 + Dy^3 + Exy.$$

Comme $x \mapsto R(x, 0) = Ax^3$ doit être une permutation, nous déduisons que $A \neq 0$. Une multiplication par une constante non-nulle ne change ni le degré ni la propriété intégrale. De ce fait, nous considérons un cas normalisé où $A = 1$. Il nous faut que $x \mapsto R(x, y)$ soit une permutation pour n'importe quel y. Nous sommes en caractéristique 2 et, en utilisant les notations du théorème 2.21, nous avons toujours que $q \equiv 2 \mod 3$ car n est impair. Ainsi, pour vérifier la condition intégrale , le théorème 2.21 impose que $(By)^2 = Cy^2 + Ey$ pour tout y. Ceci implique que $E = 0$ et $B^2 = C$. Le polynôme peut donc être écrit $R(x, y) = x^3+Bx^2y+B^2xy^2+Dy^3$ que nous factorisons en $R(x, y) = (x+By)^3+(B^3+D)y^3$. Prendre $\alpha = B$ et $\beta = B^3 + D$ nous donne le lemme. □

Remarque 2.3. Le lemme 2.10 exclut les termes en x^2, y^2, x et y de R. Comme ces termes sont de degré algébrique 1, il pourraient être ajoutés sans changer ni la linéarité ni les propriétés différentielles de V_R, ce pour quoi nous les ignorons.

Dans ce contexte, les résultats de [PUB16] peuvent être interprétés comme l'étude du cas particulier $\beta = 1$. Si $\alpha = 1$, les Papillons ouverts et fermés sont fonctionnellement équivalents aux fonctions des figures 2.13a et 2.13b.

2.3.3 Relations d'équivalence

Lemme 2.9. *La permutation H_R et la fonction V_R de $(\mathbb{F}_{2^n})^2$ sont équivalentes CCZ.*

Démonstration. La preuve est identique à celle du lemme 2 dans [PUB16]. □

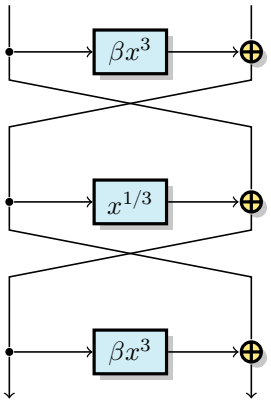


Figure 2.13(a): Papillon ouvert généralisé pour $\alpha = 1$ (Feistel).

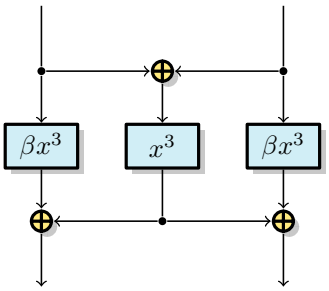


Figure 2.13(b): Papillon fermé généralisé pour $\alpha = 1$ (Lai-Massey).

Le lemme 2.9 établit qu'un Papillon ouvert et un Papillon fermé de mêmes paramètres sont équivalents CCZ. Il existe d'autres relations d'équivalence entre les Papillons.

— Si l'exposant est de la forme $e = 3 \times 2^t$, alors le Papillon fermé correspondant est équivalent affine au Papillon fermé avec les mêmes α, β mais un exposant $e = 3$. De ce fait, les résultats valent aussi pour

$$R(x, y) = (x + \alpha y)^{3\times 2^t} + \beta y^{3\times 2^t}$$

quel que soit t.

— Les Papillons fermés $\mathsf{V}_{\alpha,\beta}$ et $\mathsf{V}_{\alpha^2,\beta^2}$ sont équivalents affines puisque

$$\mathsf{V}_{\alpha^2,\beta^2}(x^2, y^2) = \left(\mathsf{V}_{\alpha,\beta}(x, y)\right)^2 .$$

— Pour tout $\alpha \neq 1$, les Papillons fermés $\mathsf{V}_{\alpha,\beta}$ et $\mathsf{V}_{\alpha,\beta'}$ avec $\beta' = \beta^{-1}(1 + \alpha)^6$ sont équivalents affine.
Cette équivalence est obtenue en composant $\mathsf{V}_{\alpha,\beta}$ avec l'inverse de la permutation linéaire

$$L : (x, y) \mapsto (z_1, z_2) = (\alpha x + y, x + \alpha y) .$$

En effet,

$$\begin{aligned}
&\mathsf{V}_{\alpha,\beta} \circ L^{-1}(x, y) \\
&= \left(z_2^3 + \beta\left[(1 + \alpha)^{-2}(z_1 + \alpha z_2)\right]^3, z_1^3 + \beta\left[(1 + \alpha)^{-2}(z_2 + \alpha z_1)\right]^3\right) \\
&= \left((1 + \alpha)^{-6}\left[(z_1 + \alpha z_2)^3 + \beta' z_2^3\right], (1 + \alpha)^{-6}\left[(z_2 + \alpha z_1)^3 + \beta' z_1^3\right]\right) .
\end{aligned}$$

— Soit $\otimes$ défini par $(a, b) \otimes (c, d) = (ac, bd)$ pour toutes paires (a, b) et (c, d) de $(\mathbb{F}_{2^n})^2$. Alors les Papillons généralisés exhibent la même stabilité multiplicative que ceux décrits dans [PUB16], à savoir :

$$\mathsf{V}_{\alpha,\beta}\big((\lambda, \lambda) \otimes (x, y)\big) = (\lambda^3, \lambda^3) \otimes \mathsf{V}_{\alpha,\beta}(x, y),$$

et

$$\mathsf{H}_{\alpha,\beta}\big((\lambda^3, \lambda) \otimes (x, y)\big) = (\lambda^3, \lambda) \otimes \mathsf{H}_{\alpha,\beta}(x, y).$$

Notons que ces propriétés multiplicatives correspondent à la *subspace property* (*propriété de sous-espace*) introduite dans [Bro+10] et étudiée dans [Göl15].

2.3.4 Propriétés cryptographiques

En nous reposant sur des résultats connus sur les fonctions booléennes quadratiques, nous obtenons les bornes de sécurité suivantes pour les Papillons généralisés (les preuves sont données en section 2.3.5) :

Théorème 2.20 (Sécurité d'un Papillon généralisé [CDP17])**.** *Les propriétés cryptographiques des Papillons généralisés sur $2n$ bits, n impair, $\mathsf{V}_{\alpha,\beta}$ et $\mathsf{H}_{\alpha,\beta}$, qui sont construites sur les fonctions $R : (x, y) \mapsto (x + \alpha y)^3 + \beta y^3$ avec α, $\beta \neq 0$ sont les suivantes :*

- *le degré algébrique de $\mathsf{V}_{\alpha,\beta}$ est toujours égal à 2,*
- *si $n = 3$, $\alpha \neq 0$, $\mathrm{Tr}(\alpha) = 0$ et $\beta \in \{\alpha^3 + \alpha, \alpha^3 + 1/\alpha\}$ alors les Papillons sont APN, ont une linéarité de 2^{n+1} et le degré algébrique de $\mathsf{H}_{\alpha,\beta}$ est égal à $n + 1$;*
- *si $\beta = (1 + \alpha)^3$, alors l'uniformité différentielle est égale à 2^{n+1}, la linéarité est égale à $2^{(3n+1)/2}$ et le degré algébrique de $\mathsf{H}_{\alpha,\beta}$ est égal à n ;*
- *sinon, l'uniformité différentielle est égale à 4, la linéarité est égale à 2^{n+1} et le degré algébrique de $\mathsf{H}_{\alpha,\beta}$ est n ou $n + 1$. Il vaut n si et seulement si*

$$1 + \alpha\beta + \alpha^4 = (\beta + \alpha + \alpha^3)^2.$$

En particulier, il n'existe pas de Papillon APN sur plus de 6 bits.

Les Papillons ouverts généralisés avec $\beta \neq (1+\alpha)^3$ forment une famille de permutations sur $2n$ bits, n impair, dont la linéarité et l'uniformité différentielle sont les meilleures qu'on sache atteindre. Qui plus est, la seule permutation APN sur un corps de dimension paire (la permutation de Dillon) est, à équivalence affine près, un Papillon généralisé.

Notons que nous parvenons à exhiber des critères nécessaires pour qu'un Papillon généralisé soit APN sur n'importe quelle taille. Malheureusement, cette condition implique que le Papillon généralisé est de taille 6 bits (et donc équivalent CCZ à la boîte-S de Dillon [12]).

Ce résultat négatif est à l'évidence décevant, mais au moins cette question ouverte difficile est désormais close : il n'existe pas d'autres Papillons APN que ceux sur 6 bits (du moins pas de Papillon avec e de la forme 3×2^i généralisé de cette manière).

Deux articles de ToSC en 2017 et 2018 portent sur des généralisations supplémentaires – avec des résultats similaires. Il s'agit d'un article de Fu, Feng et Wu [FFW17] qui étudie les Papillons avec $\beta = 1$ et des exposants de la forme $2^i + 1$ quand $pgcd(i, n) = 1$ et d'un article de Li, Tian, Yu et Wang [Li+18] sur les Papillons pour tout β et pour tous les exposants de la forme $2^i + 1$.

2.3.5 Papillons : preuves

À la différence des preuves sur les réseaux de Feistel et les réseaux de type MISTY, les preuves sur les Papillons ne reposent pas sur l'étude de quelques différentielles particulières (resp. masques linéaires particuliers). En effet, étudier des différentielles particulières (resp. masques particuliers) est suffisant pour donner une borne inférieure sur l'uniformité différentielle (resp. la linéarité). Ici, nous sommes intéressés par des bornes supérieures, il nous faut donc au choix : trouver des différentielles dont on peut prouver qu'elles sont toujours les meilleures (ce qui est extrêmement complexe et potentiellement impossible) ou prouver des bornes qui soient vraies sur toute différentielle. Ceci explique que nous nous intéresserons à des différentielles moins structurées et moins simples, et que les preuves seront donc moins directes que dans le cas des réseaux de Feistel et des réseaux de type MISTY.

12. Par recherche exhaustive sur les Papillons généralisés sur 6 bits, tous sont équivalents CCZ à la boîte-S de Dillon.

2.3.5.1 Forme de R

En premier lieu, nous prouvons que notre généralisation revient à étudier les R de la forme $R(x, y) = (x + \alpha y)^3 + \beta y^3$.

Lemme 2.10 (Restriction sur le degré)**.** *Soit R un polynôme bivarié de $\mathbb{F}_{2^n}$ tel que $R_y : x \mapsto R(x, y)$ soit une permutation pour tout y et tel que tous les termes dans R soient de degré au plus 3. Alors R peut être caractérisé en utilisant deux éléments de $\mathbb{F}_{2^n}$ notés α et β par*

$$R(x, y) = (x + \alpha y)^3 + \beta y^3 \ .$$

Nous noterons les Papillons basés sur de tels polynômes R par $\mathsf{V}_{\alpha,\beta}$ et $\mathsf{H}_{\alpha,\beta}$ pour les Papillons fermé et ouvert respectivement.

La preuve de ce lemme repose sur le théorème suivant.

Théorème 2.21 (Corollaire 2.9 dans [MS87])**.** *Soit $\mathbb{F}_q$ de caractéristique différente de 3. Alors $f : x \mapsto ax^3 + bx^2 + cx + d$ $(a \neq 0)$ permute $\mathbb{F}_q$ si et seulement si $b^2 = 3ac$ et $q \equiv 2 \mod 3$.*

Preuve du lemme 2.10. Soit

$$R(x, y) = Ax^3 + Bx^2y + Cxy^2 + Dy^3 + Exy.$$

Comme $x \mapsto R(x, 0) = Ax^3$ doit être une permutation, nous déduisons que $A \neq 0$. Une multiplication par une constante non-nulle ne change ni le degré ni la propriété intégrale. Ainsi, nous pouvons nous contenter du cas normalisé où $A = 1$. Il nous faut que $x \mapsto R(x, y)$ soit une permutation pour tout y. Nous sommes en caractéristique 2 et, en utilisant les notations du théorème 2.21, il est toujours vrai que $q \equiv 2 \mod 3$ puisque n est impair. De ce fait, pour vérifier la condition, le théorème 2.21 impose que $(By)^2 = Cy^2 + Ey$ pour tout y. Ceci implique que $E = 0$ et $B^2 = C$. Le polynôme peut donc s'écrire sous la forme $R(x, y) = x^3 + Bx^2y + B^2xy^2 + Dy^3$ que nous factorisons en $R(x, y) = (x + By)^3 + (B^3 + D)y^3$. En posant simplement $\alpha = B$ et $\beta = B^3 + D$, nous obtenons le lemme. □

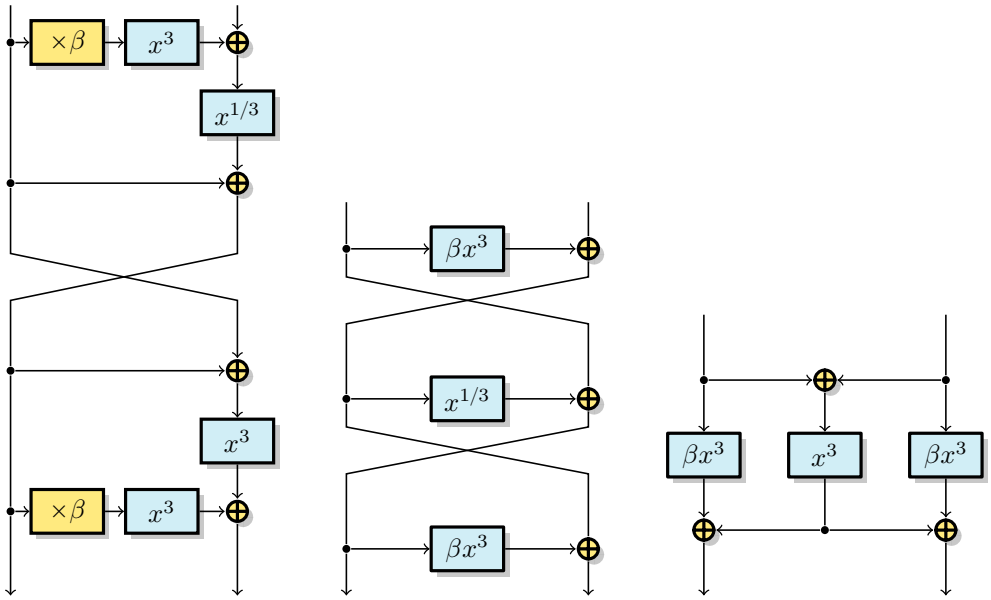
Remarque 2.4. Le lemme 2.10 exclut les termes en x^2, y^2, x et y de R. Comme ces termes sont de degré algébrique 1, on pourrait les ajouter sans modifier la linéarité et les propriétés différentielles de V_R, ce qui explique que nous les ignorons.

Dans ce contexte, les résultats dans [PUB16] peuvent être interprétés comme la manipulation du cas particulier $\beta = 1$. Si $\alpha = 1$, les Papillons ouvert et fermé sont fonctionnellement équivalents aux fonctions présentées en figure 2.14.

2.3.5.2 Calcul de la linéarité des Papillons généralisés

L'élément clef des preuves de cette section est la restriction aux fonctions R quadratiques, qui implique que le Papillon fermé est lui aussi quadratique. En effet, dans ce cas-là, le spectre de Walsh peut être relativement aisément calculé à partir de propriétés des dérivées. Dans un premier temps, nous détaillons le principe général, puis nous appliquons cette méthode au cas particulier des Papillons fermés.

Méthode générale pour calculer la linéarité. Comme le Papillon fermé $\mathsf{V}_{\alpha,\beta}$ a un degré algébrique de 2, sa linéarité peut être évaluée en calculant le nombre de structures linéaires de ses composantes, i.e., le nombre de dérivées constantes des composantes de la fonction. Cette relation est décrite dans la proposition suivante. Bien que ce résultat soit bien connu dans le domaine des fonctions booléennes (voir e.g. [MS77, chapitre 15], [Car10, proposition 15] ou [Can+01b, appendice A]), nous donnons une preuve par souci d'exhaustivité.



(a) Papillon ouvert $\mathsf{H}_{1,\beta}$. **(b)** F_β (à noter : $\mathsf{F}_\beta = \mathsf{H}_{1,\beta}$). **(c)** Papillon fermé $\mathsf{V}_{1,\beta}$.

Figure 2.14 – L'équivalence entre $\mathsf{H}_{1,\beta}$ et F_β.

Proposition 2.6. *Soit f une fonction booléenne quadratique de n variables,* i.e. $F : \mathbb{F}_2^n \to \mathbb{F}_2$. *Soit $\mathsf{LS}(f)$ l'espace linéaire de f, i.e.,*

$$\mathsf{LS}(f) = \{a \in \mathbb{F}_2^n : D_a f(x) = \varepsilon, \ \ \forall x \in \mathbb{F}_2^n\}$$

où $\varepsilon \in \{0,1\}$. Alors $s = \dim \mathsf{LS}(f)$ a la même parité que n et $\mathcal{L}(f) = 2^{\frac{n+s}{2}}$. De plus, les coefficients de Walsh de f prennent 2^{n-s} fois la valeur $\pm 2^{\frac{n+s}{2}}$ et $(2^n - 2^{n-s})$ fois la valeur 0.

Démonstration.
Tout d'abord, il est clair que $\mathsf{LS}(f)$ est un sous-espace linéaire de $\mathbb{F}_2^n$. De plus, puisque

$$D_a f(x) + D_b f(x) = D_{a+b} f(x + a),$$

seules deux situations peuvent se produire. Soit $D_a f = 0$ pour tout $a \in \mathsf{LS}(f)$, soit $D_a f = 0$ pour exactement la moitié des éléments dans $\mathsf{LS}(f)$.

Dans le second cas, $\mathsf{LS}_1(f) = \{a : D_a f = 1\}$ est un translaté de $\mathsf{LS}_0(f) = \{a : D_a f = 0\}$. Alors nous utilisons le fait que les coefficients de Walsh de f sont liés au poids de ses dérivées (voir e.g. [Can+00, Lemma 1]), plus précisément

$$\left(\widehat{f}(u)\right)^2 = 2^n \sum_{a \in \mathbb{F}_2^n} (-1)^{u \cdot a} \widehat{D_a f}(0) \ .$$

Étant donné que f est de degré algébrique 2, $D_a f$ est de degré algébrique 1 ou est constante. Si son degré est 1, c'est qu'elle est équilibrée, i.e., $\widehat{D_a f}(0) = 0$. Il s'ensuit que

$$\widehat{f}(u)^2 = 2^{2n} \left(\sum_{a \in \mathsf{LS}_0(f)} (-1)^{u \cdot a} - \sum_{a \in \mathsf{LS}_1(f)} (-1)^{u \cdot a} \right) .$$

Si $\mathsf{LS}_1(f) = \varnothing$,

$$\widehat{f}(u)^2 \in \{0, 2^{2n+\dim \mathsf{LS}(f)}\} .$$

Sinon, $\mathsf{LS}_1(f) = b + \mathsf{LS}_0(f)$ pour un certain b, ce qui implique que

$$\widehat{f}(u)^2 = 2^{2n}\left(1 + (-1)^{u\cdot b}\right)\left(\sum_{a\in \mathsf{LS}_0(f)} (-1)^{u\cdot a}\right) .$$

Alors

$$\widehat{f}(u)^2 \in \{0, 2^{2n+\dim \mathsf{LS}(f)}\} .$$

Le nombre d'occurrences des valeurs 0 et $\mathcal{L}(f)$ dans le spectre de Walsh se déduit directement de la relation de Parseval. □

Nous utiliserons aussi le lemme suivant, qui est un cas particulier simple du résultat donné dans [Bra+07, corollaire 1].

Lemme 2.11. *Soit n en entier impair et a, b, c trois éléments de $\mathbb{F}_{2^n}$ non-nuls. Alors l'équation*

$$aX^{16} + bX^4 + cX = 0$$

possède 1, 2 ou 4 solutions dans $\mathbb{F}_{2^n}$.

Démonstration.
Soit $P(X) = aX^{16} + bX^4 + cX$. Comme P est un polynôme linéarisé non-nul de degré au plus 16, ses racines dans $\mathbb{F}_{2^n}$ forment un espace linéaire de $\mathbb{F}_{2^n}$, vu comme espace vectoriel sur $\mathbb{F}_2$.

Alors P possède 2^r racines dans $\mathbb{F}_{2^n}$, avec $0 \leqslant r \leqslant 4$.

Puisque $\mathbb{F}_{2^n}$ est un sous-corps de $\mathbb{F}_{2^{2n}}$, les racines de P dans $\mathbb{F}_{2^n}$ sont aussi incluses dans l'ensemble de toutes les racines de P dans $\mathbb{F}_{2^{2n}}$.

Soit $\beta \in \mathbb{F}_{4^n}$ tel que $\beta^2 + \beta + 1 = 0$. Alors, pour tout $x \in \mathbb{F}_{2^n}$ tel que $P(x) = 0$, on a $P(\beta x) = 0$ et $P(\beta^2 x) = 0$, et aucune de ces deux autres racines ne se trouve dans $\mathbb{F}_{2^n}$ car n est impair.

De plus, si x et x' sont deux racines distinctes de P dans $\mathbb{F}_{2^n}$, alors $\{x, \beta x, \beta^2 x, x', \beta x', \beta^2 x'\}$ sont six racines distinctes de P dans $\mathbb{F}_{4^n}$. Ceci implique que, si P possède sept racines non-nulles dans $\mathbb{F}_{2^n}$, c'est qu'il aurait plus de 16 racines dans $\mathbb{F}_{4^n}$ ce qui est impossible. De ce fait, P possède au plus 4 racines dans $\mathbb{F}_{2^n}$. □

Linéarité des Papillons généralisés.

Théorème 2.22. *Soit $n > 1$ un entier impair et (α, β) une paire d'éléments non-nuls de $\mathbb{F}_{2^n}$. Alors la linéarité de $\mathsf{V}_{\alpha,\beta}$ sur $\mathbb{F}_{2^n} \times \mathbb{F}_{2^n}$ est 2^{n+1} si $\beta \neq (1+\alpha)^3$. De plus, les coefficients de Walsh de $\mathsf{V}_{\alpha,\beta}$ sont dans $\{0, \pm 2^n, \pm 2^{n+1}\}$.*

Si $\beta = (1+\alpha)^3$, la linéarité de $\mathsf{V}_{\alpha,\beta}$ vaut $2^{\frac{3n+1}{2}}$.

Démonstration.
La linéarité de $\mathsf{V}_{\alpha,\beta}$ est par définition la linéarité maximale de ses composantes non-triviales, i.e., des fonctions booléennes

$$f_{\lambda,\mu} : (x, y) \mapsto \mathrm{Tr}(\lambda R(x, y)) + \mathrm{Tr}(\mu R(y, x))$$

pour $\lambda, \mu \in \mathbb{F}_{2^n}$, où Tr symbolise la fonction trace de $\mathbb{F}_{2^n}$ dans $\mathbb{F}_2$.

Déterminons en premier lieu la linéarité de $f_{0,\mu}$. Comme $x \mapsto R(x, y)$ est une permutation pour tout y fixé, nous déduisons que $f_{0,\mu}$ est équilibrée sur tout ensemble $(a, \mathbb{F}_{2^n})$, impliquant qu'elle est équilibrée sur $\mathbb{F}_{2^n}^2$.

La linéarité de $f_{0,\mu}$ est déterminée par le nombre de paires (a,b) telles que $D_{(a,b)}f_{0,\mu}$ soit constante.

On a

$$\begin{gathered} R(y+b,x+a)+R(y,x) \\ = \\ (\alpha^2 b + a(\alpha^3+\beta))x^2 + ((\alpha^3+\beta)a^2+\alpha b^2)x + (b+\alpha a)y^2 + (b+\alpha a)^2 y + R(b,a)\ . \end{gathered}$$

Alors,

$$\begin{aligned} D_{(a,b)}f_{0,\mu} \ = \ & \mathrm{Tr}\left(x^2\left(\mu(\alpha^2 b + a(\alpha^3+\beta)) + \mu^2((\alpha^3+\beta)^2a^4+\alpha^2 b^4)\right)\right) \\ & + \mathrm{Tr}\left(y^2\left(\mu(b+\alpha a) + \mu^2(b+\alpha a)^4\right)\right) + c, \text{ avec } c \in \mathbb{F}_2 \end{aligned}$$

ce qui induit que $D_{(a,b)}f_{0,\mu}$ est constante si et seulement si (a,b) satisfait

$$\begin{cases} \mu(\alpha^2 b + a(\alpha^3+\beta)) + \mu^2((\alpha^3+\beta)^2a^4+\alpha^2 b^4) \ = \ 0 \\ \mu(b+\alpha a) + \mu^2(b+\alpha a)^4 \ = \ 0\ . \end{cases}$$

Or $\mu \neq 0$, donc la seconde équation implique que

$$b = \alpha a + \delta \text{ avec } \delta \in \{0, \mu^{-1/3}\}\ .$$

En remplaçant la valeur de b dans la première équation, on obtient

$$\beta^2 a^4 + \beta a = \delta'$$

où δ' peut prendre deux valeurs dont 0.

Puisque $\beta \neq 0$, nous déduisons que a prend deux valeurs distinctes lorsque $\delta' = 0$, et au plus deux valeurs lorsque $\delta' \neq 0$. D'où nous déduisons que le nombre de paires (a,b) satisfaisant ces deux équations est soit 2, soit 4.

Nous savons grâce à la proposition 2.6 que la dimension de l'espace linéaire est paire, ce qui veut dire que $D_{(a,b)}f_{0,\mu}$ est constante pour quatre valeurs de (a,b). Par conséquent, $f_{0,\mu}$ a une linéarité d'exactement 2^{n+1}.

Intéressons-nous maintenant au cas où $\lambda \neq 0$. Dans ce cas, il existe $\lambda' \in \mathbb{F}_{2^n}$ tel que $\lambda = \lambda'^3$ car $x \mapsto x^3$ est une permutation de $\mathbb{F}_{2^n}$ pour tout n impair.

Il s'ensuit que, pour tout λ non-nul,

$$\lambda R(x,y) = (x\lambda' + \alpha y\lambda')^3 + \beta(y\lambda')^3 = R(x\lambda', y\lambda')\ .$$

En conséquence,

$$\begin{aligned} f_{\lambda,\mu}(x,y) \ &= \ \mathrm{Tr}\left(\lambda R(x,y) + \mu R(y,x)\right) \\ &= \ \mathrm{Tr}\left(R(x\lambda', y\lambda') + \mu\lambda^{-1}R(y\lambda', x\lambda')\right) \\ &= \ f_{1,\mu\lambda^{-1}}(x\lambda', y\lambda')\ . \end{aligned}$$

Nous en déduisons que, pour $\lambda \neq 0$, $f_{\lambda,\mu}$ est linéairement équivalent à $f_{1,\mu\lambda^{-1}}$, ce qui implique que ces deux fonctions ont la même linéarité. Il suffit donc de calculer la dimension de l'espace linéaire de la fonction quadratique $f_{1,\lambda}$ pour tout $\lambda \in \mathbb{F}_{2^n}^*$. Soit $\gamma = \alpha^3 + \beta$. Alors, pour tous a et b dans $\mathbb{F}_{2^n}$, on a

$$\begin{aligned} D_{(a,b)}f_{1,\lambda} \ = \ & \mathrm{Tr}\left[x^2\left(a+\alpha b+\lambda(\gamma a+\alpha^2 b)\right)\right] + \mathrm{Tr}\left[x\left((a+\alpha b)^2+\lambda(\gamma a^2+\alpha b^2)\right)\right] \\ + \ & \mathrm{Tr}\left[y^2\left(\alpha^2 a+\gamma b+\lambda(\alpha a+b)\right)\right] + \mathrm{Tr}\left[y\left(\alpha a^2+\gamma b^2+\lambda(\alpha a+b)^2\right)\right] \\ + \ & f_{1,\lambda}(a,b). \end{aligned}$$

Qu'on peut réécrire en

$$D_{(a,b)}f_{1,\lambda} = \mathrm{Tr}(Ax^2) + \mathrm{Tr}(By^2) + f_{1,\lambda}(a,b)$$

ce qui montre que $D_{(a,b)}f_{1,\lambda}$ est constante si et seulement si

$$\begin{cases} A = (1+\lambda\gamma)^2a^4 + (1+\lambda\gamma)a + (\lambda\alpha+\alpha^2)^2b^4 + (\alpha+\lambda\alpha^2)b = 0 \\ B = \quad (\alpha+\lambda\alpha^2)^2a^4 + (\alpha^2+\lambda\alpha)a + (\gamma+\lambda)^2b^4 + (\gamma+\lambda)b = 0 \end{cases} \tag{2.1}$$

On en déduit que

$$\begin{aligned}(\alpha+\lambda\alpha^2)^2A + (1+\lambda\gamma)^2B = {} & a(1+\lambda\gamma)\left[(\alpha+\lambda\alpha^2)^2 + (1+\lambda\gamma)(\alpha^2+\lambda\alpha)\right] \\ & + b^4\left[(\alpha+\lambda\alpha^2)(\alpha^2+\lambda\alpha) + (\gamma+\lambda)(1+\lambda\gamma)\right]^2 \\ & + b\left[(\alpha+\lambda\alpha^2)^3 + (1+\lambda\gamma)^2(\gamma+\lambda)\right] = 0\,. \end{aligned} \tag{2.2}$$

Que les coefficients de (2.2) ne disparaissent pas simultanément.

Supposons que $\lambda \neq \gamma^{-1}$ (le cas contraire sera étudié en fin de preuve). Nous démontrons tout d'abord que, si le coefficient de a dans (2.2) disparaît, alors le coefficient de b^4 ne disparaît pas, sauf si $(\beta,\lambda) = ((1+\alpha)^3, 1)$. En effet, le coefficient de a disparaît si et seulement si

$$\lambda^2(\alpha^3+\gamma) + \lambda(\alpha\gamma+1) = \lambda\left(\lambda\beta + (\alpha^4+\alpha\beta+1)\right) = 0\,. \tag{2.3}$$

Il n'y a donc qu'une seule valeur non-nulle de λ pour laquelle ce coefficient disparaît, à savoir

$$\lambda = \beta^{-1}(\alpha^4+\alpha\beta+1). \tag{2.4}$$

Le coefficient de b^4 disparaît si et seulement si

$$(\alpha+\lambda\alpha^2)(\alpha^2+\lambda\alpha) + (\gamma+\lambda)(1+\lambda\gamma) = \lambda^2\beta + \lambda(\alpha^3+\alpha^2+\alpha+1+\beta)^2 + \beta = 0\,.$$

Ainsi, ce coefficient disparaît pour la valeur de λ donnée par l'équation (2.4) si et seulement si

$$\begin{aligned}(\alpha^4+\alpha\beta+1)^2 + (\alpha^4+\alpha\beta+1)(\alpha^3+\alpha^2+\alpha+1+\beta)^2 + \beta^2 = {} & \\ \alpha\left(\beta+(1+\alpha)^3\right)^2\left(\beta+(1+\alpha)^2\alpha\right) = {} & 0\,. \end{aligned}$$

Or cette équation a deux racines : $\beta = (1+\alpha)^3$ (impliquant $\lambda = 1$, par (2.4)) et $\beta = (1+\alpha)^2\alpha$. Ce second cas n'intervient jamais puisque (2.4) entraînerait alors que

$$\lambda = \alpha^{-1} \text{ et } \gamma = \alpha,$$

ce qui contredit l'hypothèse que $\lambda \neq \gamma^{-1}$.

De ce fait, lorsque $(\beta,\lambda) \neq ((1+\alpha)^3, 1)$, l'équation (2.2) peut s'exprimer soit par

$$Ub^4 + Vb = 0 \text{ avec } U \neq 0 \text{ soit par } a = Ub^4 + Vb\,, \tag{2.5}$$

où la première situation intervient si et seulement si (2.3) est satisfaite.

Cas 1 : (2.2)$= \boldsymbol{Ub^4 + Vb = 0}$ avec $\boldsymbol{U \neq 0}$.

Alors nous obtenons au plus 2 solutions pour b, dont $b = 0$. En remplaçant b par ces deux valeurs dans la première équation de (2.1), nous obtenons que

$$(1+\lambda\gamma)^2a^4 + (1+\lambda\gamma)a = \delta$$

pour deux valeurs de δ. Du fait que $\lambda \neq \gamma^{-1}$, nous déduisons qu'il y a au plus deux solutions a pour chaque valeur de δ, ce qui induit qu'il y a au plus quatre paires (a,b) qui satisfont (2.1).

Cas 2 : (2.2)= $a = Ub^4 + Vb$.
Dans ce second cas de (2.5), nous remplaçons a par son expression dans la première équation de (2.1). Ceci entraîne que

$$\begin{aligned} & b^{16}U^4(1+\lambda\gamma)^2 \\ + \; & b^4\left[(\lambda\alpha+\alpha^2)^2 + V^4(1+\lambda\gamma)^2 + U(1+\lambda\gamma)\right] \\ + \; & b\left[(\alpha+\lambda\alpha^2) + V(1+\lambda\gamma)\right] \\ = \; & 0\,. \end{aligned}$$

Par le lemme 2.11, cette équation a au plus 4 solutions b, à moins que tous ses coefficients ne disparaissent. Ainsi le système a au plus 4 solutions excepté lorsque $U = 0$ et

$$\begin{cases} (\lambda\alpha+\alpha^2) + V^2(1+\lambda\gamma) = 0 \\ (\alpha+\lambda\alpha^2) + V(1+\lambda\gamma) = 0\,. \end{cases} \tag{2.6}$$

Ces deux relations sont satisfaites soit lorsque $V = 0$, ce qui implique que $\alpha = \lambda = 1$, soit quand

$$\begin{cases} \alpha(1+\lambda\alpha) + V(1+\lambda\gamma) = 0 \\ (\alpha+\lambda) + V(1+\lambda\alpha) = 0\,. \end{cases}$$

Pour $V \neq 0$, en multipliant la première équation par $(1+\lambda\alpha)$ et la seconde par $(1+\lambda\gamma)$, on obtient que

$$\alpha(1+\lambda\alpha)^2 + (\alpha+\lambda)(1+\lambda\gamma) = \lambda^2\beta + \lambda(\alpha^4+\alpha\beta+1) = 0$$

ce qui contredit le fait que la seconde situation de (2.5) se produit, puisque le second cas de (2.5) signifie que (2.3) n'est pas vérifiée.
Pour $V = 0$, nous déduisons que $\alpha = \lambda = 1$. Dans ce cas, (2.1) correspond à

$$\begin{cases} (1+\gamma)^2a^4 + (1+\gamma)a = 0 \\ \beta^2b^4 + \beta b = 0\,. \end{cases}$$

Du fait que $\beta = 1+\gamma$ est non-nul, nous obtenons que ce système d'équations a au plus 4 solutions (a, b), car chaque équation a au plus 2 solutions.

Le cas $\lambda = \gamma^{-1}$.
Dans ce cas, la première équation de (2.1) équivaut à

$$(\gamma^{-1}\alpha+\alpha^2)^2b^4 + (\alpha+\gamma^{-1}\alpha^2)b = 0\,.$$

Les deux coefficients de cette équation ne peuvent pas disparaître simultanément, sauf si

$$\gamma = \alpha^{-1} = \alpha$$

ce que implique que $\alpha = \gamma = 1$ ce qui est impossible puisque $\beta \neq 0$. Ainsi, au plus deux valeurs de b (dont $b = 0$) satisfont cette équation. Remplacer b par ces deux valeurs dans la seconde équation de (2.1) amène à

$$(\alpha+\lambda\alpha^2)^2a^4 + (\alpha^2+\lambda\alpha)a = \delta$$

pour au plus deux valeurs de δ (dont $\delta = 0$). À nouveau les coefficients de a^4 et de a ne peuvent pas disparaître simultanément puisque $\alpha = \gamma = 1$ est impossible. Cette équation possède au plus deux solutions pour chaque valeur de b, ce qui donne un total d'au plus 4 paires (a, b) solutions de (2.1). Nous avons donc prouvé que, à moins que $(\beta, \lambda) = ((1+\alpha)^3, 1)$, (2.1) possède au plus quatre solutions, i.e. $f_{1,\lambda}$ a au plus quatre dérivées constantes. Nous déduisons par conséquent de la proposition 2.6 que $\mathcal{L}(f_{1,\lambda}) \in \{2^n, 2^{n+1}\}$. Il s'ensuit que $\mathcal{L}(\mathsf{V}_{\alpha,\beta}) = 2^{n+1}$ puisque ceci correspond à la linéarité de $f_{0,\lambda}$.

Le cas $(\beta, \lambda) = ((1+\alpha)^3, 1)$.
Le dernier cas est lorsque $(\beta, \lambda) = ((1+\alpha)^3, 1)$. Alors $(1+\gamma) = (\alpha^2+\alpha)$. On obtient que (2.1) est équivalente à

$$(\alpha^2+\alpha)^2(a+b)^4 + (\alpha^2+\alpha)(a+b) = 0\ .$$

Nous en déduisons que $a+b$ prend exactement deux valeurs, ce qui implique qu'il y a exactement 2^{n+1} paires (a,b) solutions de (2.1). Il s'ensuit que, pour $\beta = (1+\alpha)^3$, $\mathcal{L}(\mathsf{V}_{\alpha,\beta}) = 2^{\frac{3n+1}{2}}$.

□

2.3.5.3 Uniformité différentielle des Papillons généralisés

Dans cette section, nous décrivons les propriétés différentielles des Papillons généralisés. Premièrement, la section 2.3.5.3 sert à prouver le théorème 2.23, qui montre que les Papillons généralisés $\mathsf{V}_{\alpha,\beta}$ et $\mathsf{H}_{\alpha,\beta}$ ont une uniformité différentielle d'au plus 4, à moins que $\beta = (1+\alpha)^3$. Ensuite, le théorème 2.24 et sa conséquence immédiate, le Corollaire 2.3, donnent des conditions nécessaires et suffisantes sur α et β pour qu'un Papillon généralisé soit APN. Ce corollaire est utilisée dans la proposition 2.7 qui établit qu'il n'y a pas de Papillon APN lorsque $n > 3$. Ces résultats sont présentés et prouvés en section 2.3.5.3. La section 2.3.5.3 se concentre sur le cas particulier $\alpha = \beta = 1$, pour lequel le papillon généralisé est équivalent à 3 tours d'un réseau de Feistel. Dans ce cas, nous retrouvons, avec une preuve différente, un résultat de [LW14], qui établit que la table des différences des Papillons correspondants ne contient pas de valeur 2. Enfin, nous démontrons en section 2.3.5.4 que le spectre de Walsh et la table des différences des Papillons généralisés sont entièrement déterminés par le nombre de composantes courbes du Papillon fermé.

Mais en premier lieu, nous exposons un lemme qui joue un rôle crucial dans ces preuves. Il permet de dériver aisément le nombre de solutions maximal de certaines équations de degré 4 qui apparaissent régulièrement dans notre preuve. Comme la plupart des preuves de cette section reposent sur le nombre de solutions d'équations univariées sur $\mathbb{F}_{2^n}$, la notion de degré qui sera employée ici sera toujours le degré univarié.

Lemme 2.12. *Soient U, V des éléments de $\mathbb{F}_{2^n}$ avec n impair et soit $Uz^4 + Vz^2 + (U+V)z = C$ une équation de degré 4 en z. Cette équation possède :*
- *0 ou 2^n solutions si $U = V = 0$,*
- *0 ou 4 solutions si $U \neq 0$, $U \neq V$ et $\mathrm{Tr}(V/U) = 1$,*
- *0 ou 2 solutions sinon, c'est-à-dire si l'une des conditions suivantes est vérifiée :*
  - *$U = 0, V \neq 0$,*
  - *$U \neq 0$ et $V = U$,*
  - *$U \neq 0$ et $\mathrm{Tr}(V/U) = 0$,*

Démonstration.
Tout d'abord, pour toute valeur de la constante C, le nombre de solutions de l'équation est soit 0 soit égale au nombre de solutions de l'équation linéaire $Uz^4 + Vz^2 + (U+V)z = 0$. Il suffit donc d'étudier le cas $C = 0$. À l'évidence, le nombre de solutions est toujours pair, puisque lorsque z est solution, $z+1$ l'est aussi.

Si $U = V = 0$, l'équation linéarisée n'utilise pas z, ce qui veut dire que toutes les valeurs de z la satisfont. Supposons désormais que soit $U \neq 0$, soit $V \neq 0$.

Si $U = 0$, l'équation correspond à $Vz(z+1) = 0$, ce qui implique qu'elle a 2 solutions.

Supposons maintenant que $U \neq 0$. Dans ce cas, nous pouvons réécrire l'équation en

$$Uz(z+1)\big(1 + V/U + z(z+1)\big) = 0.$$

Bien évidemment, $z = 0$ et $z = 1$ sont des solutions. En fait, ce sont les seules si $V = U$. Supposons alors que $V \neq U$. Le terme $(z^2 + z + 1 + V/U)$ peut être égal à 0 si et seulement

si $\mathrm{Tr}(V/U) = 1$, ce qui signifie que l'équation linéarisée a 2 solutions si $\mathrm{Tr}(V/U) = 0$ et 4 sinon. □

Le cas non-APN

Théorème 2.23 (Uniformité différentielle d'un Papillon généralisé)**.** *Soit $n > 1$ un entier impair et (α, β) une paire d'éléments non-nuls dans $\mathbb{F}_{2^n}$. Si $\beta \neq (1+\alpha)^3$, le Papillon généralisé de paramètres α et β a une uniformité différentielle d'au plus 4. De plus, son uniformité différentielle est* exactement *4 sauf si $\beta \in \{(\alpha + \alpha^3), (\alpha^{-1} + \alpha^3)\}$.*

Si $\beta = (1+\alpha)^3$, le Papillon généralisé de paramètres α et β a une uniformité différentielle de 2^{n+1}.

Démonstration.
Afin de borner l'uniformité différentielle de $\mathsf{V}_{\alpha,\beta}$, il nous faut borner le nombre de solutions (x, y) du système suivant :

$$\begin{cases} R(x,y) + R(x+a, y+b) = c \\ R(y,x) + R(y+b, x+a) = d \end{cases}$$

pour tout quadruplet (a, b, c, d) de $\mathbb{F}_{2^n}$ avec $(a, b) \neq (0, 0)$. On a

$$R(x,y) + R(x+a, y+b)$$
$$=$$
$$(ax^2 + a^2x) + \alpha(bx^2 + a^2y) + \alpha^2(b^2x + ay^2) + (\alpha^3 + \beta)(by^2 + b^2y) + R(a,b)\ .$$

Soit $u = a + \alpha b$. Alors

$$R(x,y) + R(x+a, y+b) = ux^2 + u^2x + (\alpha^2 u + b\beta)y^2 + (\alpha u^2 + b^2\beta)y + R(a,b)$$

Similairement, pour $v = \alpha a + b$, on a

$$R(y,x) + R(y+b, x+a) = (\alpha^2 v + a\beta)x^2 + (\alpha v^2 + a^2\beta)x + vy^2 + v^2y + R(b,a)\ ,$$

ce qui implique que nous cherchons les solutions de

$$\begin{cases} ux^2 + u^2x + (\alpha^2 u + b\beta)y^2 + (\alpha u^2 + b^2\beta)y = c' \\ (\alpha^2 v + a\beta)x^2 + (\alpha v^2 + a^2\beta)x + vy^2 + v^2y = d' \end{cases} \tag{2.7}$$

Les cas spéciaux. Concentrons-nous d'abord sur trois cas particuliers, à savoir $b = \alpha^{-1}a, \alpha a, 0$. Le reste de la preuve sera dédié à l'étude du cas général, où b diffère de ces trois valeurs. Nous considérerons aussi un quatrième cas particulier qui correspond à $\beta = (1+\alpha)^3$ et $b = a$.

— $\boldsymbol{b = \alpha^{-1}a}$, ou de manière équivalente $u = 0$. Notons que ni a, ni b ne disparaît, puisque cela impliquerait que $a = b = 0$, ce qui a été exclu. Dans ce cas, (2.7) peut se réécrire en

$$\begin{cases} (b\beta)y^2 + (b^2\beta)y = c' \\ (\alpha^2 v + a\beta)x^2 + (\alpha v^2 + a^2\beta)x + vy^2 + v^2y = d'\ . \end{cases}$$

Puisque $\beta \neq 0$ et $b \neq 0$, nous pouvons déduire que la première équation possède au plus deux solutions y_0 et y_1. Pour chacune de ces deux solutions, la seconde équation possède au plus deux solutions car les coefficients de x^2 et de x ne peuvent pas disparaître simultanément. En effet,

$$(\alpha^2 v + a\beta) = (\alpha v^2 + a^2\beta) = 0 \tag{2.8}$$

implique que

$$a^2\beta = \alpha v^2 = \alpha^2 a v \ ,$$

ce qui entraîne

$$\alpha v(v + a\alpha) = \alpha v b = 0 \ ,$$

qui est impossible étant donné que $v = 0$ combiné à (2.8) impliquerait que $a = 0$. Ainsi, (2.7) possède au plus quatre solutions lorsque $u = 0$.

— $\boldsymbol{b = \alpha a}$, ou de manière équivalente $v = 0$. Ce cas est similaire au précédent. En effet, (2.7) correspond maintenant à

$$\begin{cases} ux^2 + u^2 x + (\alpha^2 u + b\beta)y^2 + (\alpha u^2 + b^2\beta)y = c' \\ a\beta x^2 + a^2\beta x = d' \end{cases}$$

Comme $a\beta \neq 0$, la seconde équation possède au plus deux solutions x_0 et x_1. Pour chacune de ces solutions, la première équation possède au plus deux solutions pour y puisque les coefficients de y^2 et y ne peuvent pas disparaître simultanément. Autrement, on aurait

$$b^2\beta = \alpha u^2 = \alpha^2 b u$$

et donc $\alpha u a = 0$.

— $\boldsymbol{b = 0}$. Alors le système (2.7) correspond à

$$\begin{cases} ax^2 + a^2 x + \alpha^2 a y^2 + \alpha a^2 y = c' \\ (\alpha^3 a + a\beta)x^2 + (\alpha^3 a^2 + a^2\beta)x + \alpha a y^2 + \alpha^2 a^2 y = d' \ . \end{cases}$$

En sommant la première équation et la deuxième multipliée par α, on obtient que

$$y\alpha a^2(1 + \alpha^2) = (a + a\alpha^4 + a\alpha\beta)(x^2 + ax) + g$$

pour une certaine constante g. Considérons d'abord le cas où $\alpha = 1$. Alors nous avons

$$a\beta(x^2 + ax) = g \ .$$

Étant donné que $a\beta \neq 0$, cette équation possède au plus deux solutions x_0 et x_1. De plus, pour chaque x_i, la première équation dans le système donne au plus deux solutions pour y, ce qui implique au plus quatre solutions (x, y) pour le système en entier.

Supposons maintenant que $\alpha \neq 1$. Alors en remplaçant y par sa valeur, i.e. $y = \mu(x^2 + ax) + g'$, dans la première équation du système, on obtient

$$\alpha^2 a\mu^2 x^4 + \left[a + \alpha^2 a^3\mu^2 + \alpha a^2\mu\right] x^2 + \left[a^2 + \alpha a^3\mu\right] x = c' \ ,$$

où

$$\mu = \frac{(1 + \alpha^4 + \alpha\beta)}{\alpha a(1 + \alpha^2)} \ .$$

En remplaçant $x = ax'$, on déduit que

$$Ux'^4 + Vx'^2 + (U + V)x' = c' \tag{2.9}$$

avec

$$U = \alpha^2 a^5\mu^2 \text{ et } V = a^3 + \alpha^2 a^5\mu^2 + \alpha a^4\mu \ .$$

Cette équation a au plus quatre solutions x_i, et chaque x_i amène une unique solution y, ce qui implique que le système en entier a au plus quatre solutions.

Montrons maintenant que le système a au plus deux solutions pour tout $a \neq 0$ pour deux valeurs de β seulement. On notera que, comme $\mathsf{V}_{1,\beta}$ ne peut pas être APN car tout réseau de Feistel sur 3 tours a une uniformité différentielle d'au moins 4 [LW14], $\alpha = 1$ peut être exclu. Si $\mathsf{V}_{\alpha,\beta}$ est APN, alors l'équation de degré 4 précédente (2.9) possède au plus deux solutions pour tout $a \neq 0$ et tout c'. Nous dérivons du lemme 2.12 que cela arrive si et seulement si, pour tout $a \neq 0$,

$$U = 0 \text{ et } V \neq 0$$

ou

$$U = V \text{ et } U \neq 0$$

ou

$$U \neq 0 \text{ et } \mathrm{Tr}(V/U) = 0 \, .$$

Observons tout d'abord que $V \neq 0$, sinon

$$\alpha^2 a^2 \mu^2 + \alpha a \mu + 1 = 0$$

ce qui voudrait dire que $(\alpha a \mu)$ est une racine de $X^2 + X + 1$ alors que ce polynôme est irréductible sur $\mathbb{F}_{2^n}$, n impair. Alors la première condition veut dire que

$$\mu = \frac{(1 + \alpha^4 + \alpha\beta)}{\alpha a (1 + \alpha^2)} = 0$$

ou de manière équivalente

$$\beta = \alpha^{-1} + \alpha^3 \, .$$

La deuxième condition correspond à

$$\alpha a \mu = 1 \Leftrightarrow 1 + \alpha^4 + \alpha\beta = 1 + \alpha^2 \, ,$$

Cette condition équivaut à

$$\beta = \alpha + \alpha^3 \, .$$

La dernière condition correspond à

$$\begin{aligned} 0 = \mathrm{Tr}(V/U) \; &= \; \mathrm{Tr}(1) + \mathrm{Tr}\left(\frac{1 + \alpha a \mu}{a^2 \alpha^2 \mu^2}\right) \\ &= 1 + \mathrm{Tr}\left(\frac{1}{a^2 \alpha^2 \mu^2}\right) + \mathrm{Tr}\left(\frac{1}{a \alpha \mu}\right) = 1 \, , \end{aligned}$$

ce qui est impossible. De ce fait, les seules valeurs de β pour lesquelles $\mathsf{V}_{\alpha,\beta}$ peut être APN sont $\beta = \alpha^{-1} + \alpha^3$ et $\beta = \alpha + \alpha^3$.

— $\boldsymbol{b = a}$ et $\boldsymbol{\beta = (1 + \alpha)^3}$. Notons que $\beta = (1 + \alpha)^3 \neq 0$ implique que $\alpha \neq 1$. Dans ce cas, (2.7) est égal à

$$\begin{cases} b(1 + \alpha)x^2 + b^2(1 + \alpha)^2 x + b(1 + \alpha)y^2 + b^2(1 + \alpha)^2 y = c' \\ b(1 + \alpha)x^2 + b^2(1 + \alpha)^2 x + b(1 + \alpha)y^2 + b^2(1 + \alpha)^2 y = d' \, . \end{cases}$$

Il n'a donc pas de solution si $c' \neq d'$. Si $c' = d'$, ce système équivaut à la seule équation

$$(x + y)^2 + b(1 + \alpha)(x + y) = c' b^{-1} (1 + \alpha)^{-1} \, ,$$

puisque $\alpha \neq 1$ et $b \neq 0$. D'où l'on déduit que soit le système (2.7) n'a pas de solution, soit ses solutions sont de la forme $x + y = \varepsilon$ pour deux valeurs de ε, dépendantes de (b, c'). En particulier, le système (2.7) possède exactement 2^{n+1} solutions quand $c' = d' = 0$.

Le cas général. Supposons désormais que u, v et b sont tous non-nuls. Nous supposons aussi que $a = b$ et $\beta = (1+\alpha)^3$ ne sont pas vérifiées simultanément. Notons ℓ_1 et ℓ_2 les deux équations de (2.7) respectivement. Alors l'expression suivante doit être constante :

$$\begin{aligned} v\ell_1 + u\ell_2 &= \big(uv(\alpha^2+1) + au\beta\big)\, x^2 + \big(uv(u+\alpha v) + a^2 u\beta\big)\, x \\ &\quad + \big(uv(\alpha^2+1) + bv\beta\big)\, y^2 + \big(uv(\alpha u + v) + b^2 v\beta\big)\, y \\ &= \big(uv(\alpha^2+1) + au\beta\big)\,(x^2 + ax) \\ &\quad + \big(uv(\alpha^2+1) + bv\beta\big)\,(y^2 + by)\ , \end{aligned}$$

où la dernière égalité vient du fait que $(u+\alpha v) = a(\alpha^2+1)$ et $(\alpha u + v) = b(\alpha^2+1)$. Nous avons obtenu une relation de la forme

$$\lambda_0(x^2 + ax) + \lambda_1(y^2 + by) = \varepsilon \tag{2.10}$$

pour une certaine constante ε. Prouvons tout d'abord que λ_0 et λ_1 ne peuvent disparaître simultanément. Considérons d'abord le cas $\alpha = 1$. Alors

$$\lambda_0 = (a+b)a\beta \text{ et } \lambda_1 = (a+b)b\beta\ .$$

Puisque $u = a + b \neq 0$, $b \neq 0$ et $\beta \neq 0$, λ_1 ne disparaît pas.
Supposons maintenant que $\alpha \neq 1$. Alors nous pouvons réécrire

$$\beta = (1+\alpha^2)(\alpha + \beta')\ .$$

Il s'ensuit que

$$\begin{aligned} \lambda_0 &= uv(\alpha^2+1) + au\beta = u(\alpha^2+1)(b + a\beta') \\ \lambda_1 &= uv(\alpha^2+1) + bv\beta = v(\alpha^2+1)(a + b\beta')\ . \end{aligned}$$

Alors (2.10) est vérifiée avec

$$\lambda_0 = u(b + a\beta') \text{ et } \lambda_1 = v(a + b\beta')\ .$$

Ces deux coefficients ne peuvent pas disparaître simultanément : dans le cas contraire, on obtiendrait

$$a\beta' = b \text{ et } b\beta' = a$$

ce qui impliquerait que

$$ab\beta' = a^2 = b^2 \text{ et } \beta' = 1\ ,$$

ce qui a été exclu puisque cela induirait que $\beta = (1+\alpha)^3$ et $a = b$.
Combinons maintenant (2.10) avec l'une des équations dans (2.7). Il nous faut considérer deux cas :

— Si $\lambda_0 = 0$, alors (2.10), qui peut s'écrire

$$y^2 + by = \varepsilon'\ ,$$

possède au plus deux solutions y_0 et y_1. Remplacer y par ces deux valeurs dans la première équation dans (2.7) donne au plus deux solutions pour x pour chaque y_i puisque $u \neq 0$.

— Si $\lambda_0 \neq 0$, alors (2.10) peut s'écrire

$$x^2 = ax + \lambda_0^{-1}\lambda_1(y^2 + by) + \varepsilon'\ .$$

Nous remplaçons x^2 par son expression dans la première équation de (2.7) et nous obtenons que

$$(ua + u^2)x + (u\lambda_0^{-1}\lambda_1 + \alpha^2 u + b\beta)y^2 + (u\lambda_0^{-1}\lambda_1 b + \alpha u^2 + b^2\beta)y = c' .$$

Le coefficient de x ne disparaît pas puisque $u = a$ est équivalent à $b = 0$. Nous pouvons donc écrire x comme un polynôme de degré 2 en y, i.e.

$$x = \mu_2 y^2 + \mu_1 y + \mu_0 . \tag{2.11}$$

En remplaçant x par sa valeur dans(2.10), nous obtenons que

$$\lambda_1(y^2 + by) = \lambda_0 \left(\mu_2^2 y^4 + \mu_1^2 y^2 + a\mu_2 y^2 + a\mu_1 y\right) + \varepsilon''$$

ce qui entraîne

$$\lambda_0\mu_2^2 y^4 + \left(\lambda_1 + \lambda_0\mu_1^2 + \lambda_0 a\mu_2\right) y^2 + \left(\lambda_1 b + \lambda_0 a\mu_1\right) y + \varepsilon'' . \tag{2.12}$$

Supposons en premier lieu que les trois coefficients de y^4, y^2 et y ne disparaissent pas simultanément. Alors (2.12) possède au plus quatre solutions, y_i, $0 \leqslant i < 4$. De plus, nous savons par (2.11) que x est entièrement déterminé par y. Il s'ensuit que le système (2.7) possède au plus quatre solutions (x, y).
Supposons maintenant que tous les coefficients de (2.12) disparaissent. Alors nous avons $\mu_2 = 0$ et

$$\lambda_1 + \lambda_0\mu_1^2 = 0 \text{ et } \lambda_1 b + \lambda_0 a\mu_1 = 0 .$$

Ceci peut se produire dans l'une des deux situations suivantes :

— $\mu_1 = 0$ et $\lambda_1 = 0$. En utilisant le fait que λ_1 ne peut disparaître que si $\alpha \neq 1$, les définitions de λ_1 et μ_1 impliquent que

$$\alpha u^2 + b^2\beta = 0 \text{ et } u(\alpha^2 + 1) + b\beta = 0 ,$$

ce qui entraîne que

$$\alpha u^2 = u(\alpha^2 + 1)b$$

i.e.,

$$\alpha a + b = v = 0$$

ce qui a été exclu.

— $b\mu_1 = a$ et $b^2\lambda_1 = a^2\lambda_0$. Par définition de μ_2, on a que $\mu_2 = 0$ en plus de cette dernière relation implique que

$$\begin{aligned} 0 &= u\lambda_0^{-1}\lambda_1 + \alpha^2 u + b\beta \\ &= b^{-2}\left(ua^2 + u\alpha^2 b^2 + b^3\beta\right) \\ &= b^{-2}(u^3 + b^3\beta) \end{aligned}$$

i.e.,

$$\beta = (ub^{-1})^3 . \tag{2.13}$$

Comme $\mu_2 = 0$ et $\mu_1 = ab^{-1}$, (2.11) peut s'écrire

$$ay = bx + \mu_0' .$$

En remplaçant ay par sa valeur dans la deuxième équation de (2.7) multipliée par a^2, on obtient

$$\begin{aligned} x^2\left[vb^2 + \alpha^2 a^2 v + a^3\beta\right] + x\left[v^2 ab + \alpha a^2 v^2 + a^4\beta\right] &= d'' \\ \Leftrightarrow (v^3 + a^3\beta)(x^2 + ax) &= d'' . \end{aligned} \tag{2.14}$$

Les coefficients de cette équation ne disparaissent pas. Dans le cas contraire, en utilisant (2.13), on aurait

$$v^3 + a^3\beta = (\alpha a + b)^3 + (a^2 b^{-1} + \alpha a)^3 = 0$$

ce qui impliquerait que

$$b = a^2 b^{-1}$$

i.e., $a = b$ et $\beta = (\alpha + 1)^3$ ce qui a été exclu.
On en déduit que (2.14) possède au plus deux solutions x_0 et x_1. Comme y est entièrement déterminé par x (ou constante), on obtient que le système (2.7) possède au plus deux solutions dans ce cas.

□

Des Papillons APN En utilisant le théorème précédent, nous obtenons tout d'abord une condition nécessaire et suffisante pour qu'un Papillon généralisé soit APN dans le théorème 2.24. Ensuite, nous simplifions ces conditions dans le Corollaire 2.3. Finalement, nous montrons dans la proposition 2.7 que ces conditions ne peuvent être vérifiées que pour $n = 3$.

Théorème 2.24 (condition APN)**.** *Soit $\alpha \neq 0, 1$. Un Papillon généralisé de paramètres α et β est APN si et seulement si :*

$$\beta \in \{\alpha + \alpha^3, \alpha^{-1} + \alpha^3\} \text{ et } \mathrm{Tr}\left(\mathcal{A}_\alpha(e)\right) = 1, \ \forall\ e \notin \{0, \alpha, 1/\alpha\}\ ,$$

où

$$\mathcal{A}_\alpha(e) = \frac{e\alpha(1+\alpha)^2}{(1+\alpha e)(\alpha+e)^2}\ .$$

Démonstration.
Comme nous avons prouvé dans la section 2.3.3 que les Papillons généralisés de paramètres (α, β_0) et (α, β_1) où $\beta_1 = \beta_0^{-1}(1+\alpha)^6$ sont équivalents affine, il ne faut prouver le résultat que pour $\beta = \alpha + \alpha^3$. Comme précédemment, il nous faut compter le nombre de solutions de

$$\begin{cases} R(x,y) + R(x+a, y+b) = c \\ R(y,x) + R(y+b, x+a) = d \end{cases} \tag{2.15}$$

pour tout quadruplet (a, b, c, d) de $\mathbb{F}_{2^n}$ avec $(a, b) \neq (0, 0)$. Ce système est équivalent à

$$\begin{cases} ax^2 + a^2x + \alpha(bx^2 + a^2y) + \alpha^2(b^2x + ay^2) + (\alpha^3 + \beta)(by^2 + b^2y) = c_0 \\ by^2 + b^2y + \alpha(ay^2 + b^2x) + \alpha^2(a^2y + bx^2) + (\alpha^3 + \beta)(ax^2 + a^2x) = d_0\ . \end{cases}$$

Comme $\alpha \neq 1$, nous pouvons remplacer les lignes ℓ_1 et ℓ_2 de ce système par $\ell_1 + \alpha\ell_2$ et $\alpha\ell_1 + \ell_2$ pour obtenir un système avec exactement le même nombre de solutions. Nous obtenons

$$(ax^2 + a^2x)(1 + \alpha\beta + \alpha^4) + (\alpha + \alpha^3)(bx^2 + a^2y) + (\alpha^3 + \alpha + \beta)(by^2 + b^2y) = c_0$$

et

$$(by^2 + b^2y)(1 + \alpha\beta + \alpha^4) + (\alpha + \alpha^3)(ay^2 + b^2x) + (\alpha^3 + \alpha + \beta)(ax^2 + a^2x) = d_0.$$

Pour $\beta = \alpha + \alpha^3$, le système se simplifie encore en utilisant que $1 + \alpha\beta + \alpha^4 = (1 + \alpha^2)$ et $\alpha + \alpha^3 + \beta = 0$:

$$\begin{cases} (ax^2 + a^2x) + \alpha(bx^2 + a^2y) = c_1 \\ (by^2 + b^2y) + \alpha(ay^2 + b^2x) = d_1\ . \end{cases} \tag{2.16}$$

Considérons d'abord les cas $a = 0$ et $b = 0$. Rappelons que $a = b = 0$ est exclu. Si $a = 0$, alors la première ligne du système équivaut à

$$x = \left(\frac{c_1}{\alpha b}\right)^{2^{n-1}} .$$

Remplacer x par sa valeur dans la seconde ligne du système (2.16) donne une équation de degré 2 en y à coefficients non-nuls puisque $b \neq 0$, ce qui implique que (2.16) possède au plus deux solutions (x, y). Le cas $b = 0$ est similaire.

Supposons maintenant que $a \neq 0$ et $b \neq 0$, ce qui nous permet de fixer $x = ax'$ et $y = by'$. Dans ce contexte, le système (2.16) possède autant de solutions que

$$\begin{cases} a^3(x'^2 + x') + \alpha a^2 b(x'^2 + y') = c_1 \\ b^3(y'^2 + y') + \alpha a b^2(y'^2 + x') = d_1 \end{cases},$$

que nous réécrivons en utilisant que $e = a/b$ en

$$\begin{cases} e(x'^2 + x') + \alpha(x'^2 + y') = c_2 \\ e^{-1}(y'^2 + y') + \alpha(y'^2 + x') = d_2 \end{cases}. \tag{2.17}$$

Sommer ses lignes donne

$$(x'^2 + x')(e + \alpha) + (y'^2 + y')(e^{-1} + \alpha) \;=\; c_2 + d_2 .$$

Si $e = \alpha$, alors y' est fixé à y'_0 ou y'_1 avec $y'_0 + y'_1 = 1$. La première ligne du système implique dans ce cas que $x' = y'_i + c_2/\alpha$ puisque les termes en x^2 s'annulent mutuellement, ce qui signifie que le système possède au plus deux solutions. Le cas $e = \alpha^{-1}$ est similaire. Nous supposons que $e \neq \alpha, \alpha^{-1}$.

La première ligne du système (2.17) nous permet d'exprimer y' en fonction de x' :

$$y' = x'^2\left(\frac{e}{\alpha} + 1\right) + x'\frac{e}{\alpha} + \frac{c_2}{\alpha} .$$

Nous remplaçons y' par son expression dans la seconde ligne de (2.17) et obtenons

$$\begin{aligned} & y'^2\left(e^{-1} + \alpha\right) + y'e^{-1} + \alpha x' \\ &= \left(x'^2\left(\frac{e}{\alpha} + 1\right) + x'\frac{e}{\alpha}\right)^2\left(e^{-1} + \alpha\right) + \left(x'^2\left(\frac{e}{\alpha} + 1\right) + x'\frac{e}{\alpha}\right)e^{-1} + \alpha x' \\ &= x'^4\left(1 + \frac{e}{\alpha}\right)^2(\alpha + e^{-1}) + x'^2\left(\frac{e^2}{\alpha^2}(e^{-1} + \alpha) + \left(\frac{e}{\alpha} + 1\right)e^{-1}\right) \\ &\quad + x'\left(\frac{1}{\alpha} + \alpha\right) \\ &= d_3 \end{aligned}$$

pour une certaine constante d_3. Si nous posons $U = (1+e/\alpha)^2(\alpha+1/e)$ et $V = U+1/\alpha+\alpha$, alors le nombre de solutions de cette équation peut être calculé en utilisant le lemme 2.12. On a $U \neq 0$ et $U + V \neq 0$ puisque $\alpha \neq 1$. Ainsi, le nombre de solutions possibles est au plus 4 et est donné par la trace de V/U : si $\mathrm{Tr}\,(V/U) = 0$, alors l'équation a au plus 2 solutions, sinon elle a 0 ou 4 solutions. On a

$$\begin{aligned} \frac{V}{U} &= 1 + \frac{\alpha^{-1} + \alpha}{(e^{-1} + \alpha)(1 + e\alpha^{-1})^2} \\ &= 1 + \frac{e\alpha(1 + \alpha)^2}{(1 + \alpha e)(\alpha + e)^2} \end{aligned}$$

donc la fonction est APN si et seulement si

$$\mathrm{Tr}\left(\mathcal{A}_\alpha(e)\right) = 1,\ \forall e \neq 0, \alpha, 1/\alpha,\ \text{avec } \mathcal{A}_\alpha(e) = \frac{e\alpha(1+\alpha)^2}{(1+\alpha e)(\alpha+e)^2}\ .$$

□

La condition donnée par le théorème 2.24 est suffisante pour décrire tous les Papillons généralisés APN mais elle peut être grandement simplifiée. Nous commençons cette simplification dans le corollaire suivant.

Corollaire 2.3. *Soit $\alpha \neq 1$, $\beta_0 = \alpha^3 + \alpha$ et $\beta_1 = \alpha^3 + 1/\alpha$. Un Papillon généralisé de paramètres α et β est APN si et seulement si $\beta = \beta_0$ ou β_1 et*

$$\mathrm{Tr}\left(\mathcal{C}_\alpha(u)\right) = 1,\ \forall\ v \notin \left\{0, 1, 1/(1+\alpha^{-2})\right\}\ .$$

avec

$$\mathcal{C}_\alpha(u) = \left(\frac{1}{1+\alpha^{-1}}\right)^4 \frac{1}{v+v^3}.$$

Démonstration.
Nous savons par le théorème 2.24 qu'un Papillon généralisé de paramètres α et β est APN si et seulement si $\beta \in \{\beta_0, \beta_1\}$ et $\mathrm{Tr}\left(\mathcal{A}_\alpha(e)\right) = 1$ pour tout e n'appartenant pas à $\{0, \alpha, 1/\alpha\}$. Supposons que $\alpha \neq 1$ et posons $\ell = (e+\alpha)(1+\alpha)^2$. Alors nous pouvons réécrire certaines des expressions impliquées dans $\mathcal{A}_\alpha(e)$ de la façon suivante :

$$e(1+\alpha)^2 = \ell + \alpha + \alpha^3 \text{ et } (1+\alpha e)(1+\alpha)^2 = \alpha\left(\ell + \frac{(1+\alpha)^4}{\alpha}\right)\ .$$

Rappelons que $\beta_0 = \alpha + \alpha^3$ et $\beta_1 = (\alpha+1)^4/\alpha$, donc nous pouvons réécrire :

$$\begin{aligned}
\mathcal{A}_\alpha(e) &= \frac{e\alpha(1+\alpha)^2}{(1+\alpha e)(\alpha+e)^2} \\
&= \frac{\alpha\ e(1+\alpha^2)}{(1+\alpha e)(1+\alpha^2)\frac{((\alpha+e)(1+\alpha)^2)^2}{(1+\alpha)^6}} \\
&= (1+\alpha)^6 \frac{\alpha(\ell+\beta_0)}{\alpha(\ell+\beta_1)\ell^2} \\
&= \frac{\beta_0\beta_1}{\ell^2}\frac{\ell+\beta_0}{\ell+\beta_1}\ .
\end{aligned}$$

Soit $\mathcal{B}_\alpha(v) = v^2(v+1)/(v+\beta_0/\beta_1)$. Alors l'équation suivante est vérifiée :

$$\mathcal{B}_\alpha\left(\frac{\beta_0}{\ell}\right) = \frac{\beta_0\beta_1}{\ell^2}\frac{\ell+\beta_0}{\ell+\beta_1} = \mathcal{A}_\alpha(e)\ .$$

Il est donc suffisant d'étudier la trace de $\mathcal{B}_\alpha$. Cette condition $e \notin \{0, \alpha, \alpha^{-1}\}$ devient $\ell \notin \{\beta_0, 0, \beta_1\}$ et, de manière équivalent, $\beta_0/\ell \notin \{0, 1, \beta_0/\beta_1\}$. Par conséquent, le Papillon généralisé de paramètres α, β est APN si et seulement si $\beta = \beta_0$ ou β_1 et

$$\mathrm{Tr}\left(\mathcal{B}_\alpha(v)\right) = 1,\ \forall v \notin \left\{0, 1, \frac{\alpha^2}{1+\alpha^2}\right\}$$

puisque $\beta_0/\beta_1 = \alpha^2/(1+\alpha^2)$. Finalement, notons que la trace de $\mathcal{B}_\alpha(v)$ peut être simplifiée :

$$\begin{aligned}
\mathrm{Tr}\left(\mathcal{B}_\alpha(v)\right) &= \mathrm{Tr}\left(v^2 \frac{1+v}{v+\frac{\alpha^2}{1+\alpha^2}}\right) \\
&= \mathrm{Tr}\left(v^2 + \frac{v^2}{(1+\alpha^2)v+\alpha^2}\right) \\
&= \mathrm{Tr}\left(v + \frac{v^2}{(1+\alpha^2)v+\alpha^2}\right) \\
&= \mathrm{Tr}\left(\frac{(1+\alpha^2)v^2+\alpha^2 v+v^2}{(1+\alpha^2)v+\alpha^2}\right) \\
&= \mathrm{Tr}\left(\frac{v^2+v}{\gamma v+1}\right) ,
\end{aligned}$$

où $\gamma = 1+\alpha^{-2}$. Nous déduisons le résultat suivant :

$$\begin{aligned}
\mathrm{Tr}\left(\mathcal{B}_\alpha\left(u^{-1}\gamma^{-1}\right)\right) &= \mathrm{Tr}\left(\frac{\left(u^{-1}\gamma^{-1}\right)^2 + u^{-1}\gamma^{-1}}{u^{-1}+1}\right) \\
&= \mathrm{Tr}\left(\frac{\left(u^{-1}\gamma^{-1}\right)^2}{u^{-1}+1} + \frac{\left(u^{-1}\gamma^{-1}\right)^2}{u^{-2}+1}\right) \\
&= \mathrm{Tr}\left(\frac{(u^{-3}+u^{-2})\gamma^{-2}+u^{-2}\gamma^{-2}}{u^{-2}+1}\right) \\
&= \mathrm{Tr}\left(\frac{\gamma^{-2}}{u+u^3}\right) .
\end{aligned}$$

La condition $u^{-1}/\gamma \notin \{0,1,\gamma^{-1}\}$ est équivalente à $u \notin \{0,\gamma^{-1},1\}$, le même ensemble que précédemment. Ceci prouve le corollaire. □

Montrons maintenant que la dernière condition du Corollaire 2.3 ne peut être satisfaite que pour $n = 3$. En d'autres termes, les Papillons généralisés APN n'existent que pour $n = 3$. La preuve repose sur le lemme suivant.

Lemme 2.13. *[BRS67] L'équation cubique $x^3 + ax + b = 0$, où $a \in \mathbb{F}_{2^n}$ et $b \in \mathbb{F}_{2^n}^*$ a une unique solution dans $\mathbb{F}_{2^n}$ si et seulement si $\mathrm{Tr}(a^3/b^2) \neq \mathrm{Tr}(1)$.*

Proposition 2.7. *Soit $n > 1$ un entier impair, et $\lambda \in \mathbb{F}_{2^n}^*$. Si*

$$\mathrm{Tr}\left(\frac{\lambda^2}{x+x^3}\right) = 1, \ \forall\ x \notin \{0,1,\lambda\} \ , \tag{2.18}$$

alors $n = 3$.

Démonstration. Soit z dans $\mathbb{F}_{2^n}^*$ avec $\mathrm{Tr}(z) = 0$. Alors il existe un unique $x \in \mathbb{F}_{2^n}\backslash\mathbb{F}_2$ tel que

$$\frac{1}{x^3+x} = z \ .$$

En effet, puisque $z \neq 0$, ceci signifie de façon équivalente que

$$x^3 + x + \frac{1}{z} = 0 \ .$$

Nous savons par le lemme 2.13 que cette équation a une solution unique lorsque $\mathrm{Tr}(z^2) = \mathrm{Tr}(z) = 0$. Définissons z_λ par

$$z_\lambda = \frac{1}{\lambda^3+\lambda} \ ,$$

et

$$\mathcal{Z} = \{z \in \mathbb{F}_{2^n}^* \backslash \{z_\lambda\} : \mathrm{Tr}(z) = 0\} .$$

À l'évidence, $\mathcal{Z}$ est soit un hyperplan sans 0, soit un hyperplan sans 0 et z_λ (suivant la valeur de $\mathrm{Tr}(z_\lambda)$). Alors la condition (2.18) implique que, pour tout $z \in \mathcal{Z}$,

$$\mathrm{Tr}(\lambda^2 z) = 1 .$$

Supposons que $n \geqslant 5$. Alors, $\mathcal{Z}$ contient au moins $(2^{n-1} - 2) \geqslant 14$ éléments et il existe au moins deux éléments distincts z_0 et z_1 dans $\mathcal{Z}$ tels que $z_0 + z_1 \in \mathcal{Z}$. Donc ces deux éléments doivent vérifier

$$\mathrm{Tr}(\lambda^2 z_0) = \mathrm{Tr}(\lambda^2 z_1) = \mathrm{Tr}(\lambda^2 (z_0 + z_1)) = 1$$

ce qui est impossible puisque

$$\mathrm{Tr}(\lambda^2 (z_0 + z_1)) = \mathrm{Tr}(\lambda^2 z_0) + \mathrm{Tr}(\lambda^2 z_1)$$

Lorsque $n = 3$, la situation est différente puisque la condition peut être satisfaite lorsque $\mathcal{Z}$ ne contient que 2 éléments, i.e. lorsque $\mathrm{Tr}(z_\lambda) = 0$. □

Le cas des réseaux de Feistel $\alpha = \beta = 1$ Dans le cas où $\alpha = \beta = 1$, le Papillon généralisé est équivalent à un réseau de Feistel sur 3 tours de fonctions de tour $x \mapsto x^3$, $x \mapsto x^{1/3}$ et $x \mapsto x^3$. Le théorème 4 de [LW14] montre que, dans ce cas particulier, la table des différences des Papillons correspondants ne contient pas la valeur 2. Dit autrement, le nombre de solutions (x, y) de

$$\begin{cases} R(x, y) + R(x + a, y + b) = c \\ R(y, x) + R(y + b, x + a) = d \end{cases}$$

pour n'importe quel quadruplet (a, b, c, d) de $\mathbb{F}_{2^n}$ avec $(a, b) \neq (0, 0)$ est soit 0 soit 4. Nous donnons ici une preuve alternative de ce résultat.

Proposition 2.8. *[LW14, Théorème 4] Pour $\alpha = \beta = 1$, la table des différences des Papillons $\mathsf{V}_{1,1}$ et $\mathsf{H}_{1,1}$ ne contient que les valeurs 0 et 4.*

Démonstration. Tout comme dans la preuve du théorème 2.23, il nous faut compter le nombre de solutions du système (2.7), qui se simplifie en

$$\begin{cases} (a + b)x^2 + (a + b)^2 x + ay^2 + a^2 y = c' \\ bx^2 + b^2 x + (a + b)y^2 + (a + b)^2 y = d' . \end{cases} \tag{2.19}$$

— Si $a = 0$, la première ligne du système vaut $b(x^2 + bx) = c'$ qui a soit 0 soit 2 solutions, x_0 et x_1 (en se remémorant que a et b ne peuvent pas être nuls simultanément). La seconde ligne du système peut se réécrire en

$$b\left((x + y)^2 + b(x + y)\right) = d'$$

qui a 0 ou 2 solutions, impliquant $y \in \{x + z_0, x + z_1\}$. De ce fait, si la première ligne a deux solutions, la seconde a soit 0 soit 4 solutions. Le cas $b = 0$ est similaire.

— Si $a = b$, le système est composé de deux équations indépendantes de degré 2, l'une en x et l'autre en y. Si l'une de ces équations n'a pas de solutions, alors le système entier n'a pas de solution. Autrement, les deux équations ont deux solutions, et le système a donc 4 solutions.

— Si $ab(a+b) \neq 0$. Alors la première ligne ℓ_1 de (2.19)peut être remplacé par $b\ell_1 + (a+b)\ell_2$, ce qui donne

$$\begin{cases} ab(a+b)x + (ab+a^2+b^2)y^2 + (a^3+b^3+ab^2)y = \varepsilon \\ bx^2 + b^2x + (a+b)y^2 + (a+b)^2y = d' \ . \end{cases} \tag{2.20}$$

Nous multiplions alors la seconde ligne par $a^2b(a+b)^2$ et remplaçons $ab(a+b)x$ par la valeur donnée par la première ligne et obtenons

$$\begin{aligned} & y^4(ab+a^2+b^2)^2 \\ + \ & y^2\left(a^6+b^6+ab^5+a^5b+a^3b^3\right) \\ + \ & y\left(a^6b+a^5b^2+a^4b^3+a^3b^4+a^2b^5+ab^6\right) \\ = \ & \varepsilon' \ . \end{aligned}$$

Par un changement de variable $y' = by$, nous obtenons de façon équivalente

$$Uy'^4 + Vy'^2 + Wy' = b^{-8}\varepsilon' \tag{2.21}$$

où les coefficients U, V et W dépendent de $e = a/b$:

$$\begin{aligned} U &= e^4+e^2+1 = (e^2+e+1)^2 \\ V &= e^6+e^5+e^3+e+1 = (e^2+e+1)^3 \\ W &= e^6+e^5+e^4+e^3+e^2+e = U+V \ . \end{aligned}$$

Le lemme 2.12 s'applique alors. Clairement $U \neq 0$ puisque le polynôme X^2+X+1 n'a pas de racine dans $\mathbb{F}_{2^n}$ quand n est impair. De plus, $U \neq V$, sinon $e^2+e+1 = 1$ ce qui n'est pas possible puisque les cas $e \in \{0,1\}$ (*i.e.*, $a = 0$ ou $a = b$) ont été exclus. Alors l'équation (2.21) a 2 solutions seulement si $\mathrm{Tr}(V/U) = 0$. Mais

$$\mathrm{Tr}\left(\frac{V}{U}\right) = \mathrm{Tr}(e^2+e+1) = \mathrm{Tr}(1) = 1 \ ,$$

ce qui implique que l'équation (2.21) possède 0 ou 4 solutions y_i, et chaque y_i donne une unique valeur de x. Ainsi, le système entier a 0 ou 4 solutions.

□

2.3.5.4 Spectre de Walsh et table des différences des Papillons généralisés

Le théorème 2.22 souligne que, pour $\beta \neq (1+\alpha)^3$, les composantes non-nulles de $\mathsf{V}_{\alpha,\beta}$ sont soit courbes, soit ont leurs coefficients de Walsh dans $\{0, \pm 2^{n+1}\}$. Alors tout le multi-ensemble

$$\{|\widehat{\mathsf{H}_{\alpha,\beta}}(u,v)|, u \in \mathbb{F}_2^{2n}, v \in \mathbb{F}_2^{2n}\backslash\{0\}\} = \{|\widehat{\mathsf{V}_{\alpha,\beta}}(u,v)|, u \in \mathbb{F}_2^{2n}, v \in \mathbb{F}_2^{2n}\backslash\{0\}\}$$

est déterminé par le nombre B de composantes courbes de $\mathsf{V}_{\alpha,\beta}$. Qui plus est, il est bien connu qu'il y a une correspondance un-à-un entre la transformée de Walsh au carré d'une fonction vectorielle et sa table des différences (voir par exemple. [CV95 ; BN14], ou la propriété 1.10). Ainsi, différentes valeurs de B correspondent à des tables des différences distinctes. Comme tous les Papillons généralisés ont une uniformité différentielle d'au plus 4, la valeur de B détermine de manière équivalente le nombre d'occurrences de de 4 dans la table des différences de l'application. Cette correspondance est détaillée dans la proposition suivante, qui est une variante du corollaire 3 de [Ber+06].

Proposition 2.9. *Soit m un entier pair et F une application d'uniformité différentielle 4 sur $\mathbb{F}_2^m$ telle que ses composantes non-nulles soient courbes ou de spectre de Walsh $\{0, \pm 2^{\frac{m}{2}+1}\}$. Alors le nombre de 4 dans la table des différences de F est égal à*

$$2^{m-2}(2^m - 1) - 3 \times 2^{m-1}B$$

où B est le nombre de composantes courbes de F.

Notablement,

- *F est APN si et seulement si $B = \frac{2(2^m-1)}{3}$;*
- *la table des différences de F ne contient pas de 2 2 si et seulement si $B = 0$.*

Démonstration. Il est bien connu [CV95 ; BN14] que le carré de la transformée de Walsh de F est la transformée de Fourier de l'application

$$(a, b) \in \mathbb{F}_2^m \times \mathbb{F}_2^m \mapsto \delta(a, b) = \#\{x \in \mathbb{F}_2^m : F(x + a) + F(x) = b\}.$$

Une conséquence directe est alors le résultat suivant, mentionné dans [TGZ16, théorème 2] par exemple :

$$\sum_{\lambda \in \mathbb{F}_2^m} \sum_{\mu \in \mathbb{F}_2^m} \widehat{F}(\lambda, \mu)^4 = 2^{2m} \sum_{a,b \in \mathbb{F}_2^m} \delta(a, b)^2 \ .$$

De plus, nous avons

$$\sum_{\lambda \in \mathbb{F}_2^m} \widehat{F}(\lambda, \mu)^4 = \begin{cases} 2^{4m} \text{ si } \mu = 0 \\ 2^{3m} \text{ si } F_\mu \text{ est courbe} \\ 2^{3m+2} \text{ sinon.} \end{cases}$$

Il s'ensuit que

$$\sum_{\lambda \in \mathbb{F}_2^m} \sum_{\mu \in \mathbb{F}_2^m} \widehat{F}(\lambda, \mu)^4 = 2^{4m} + 2^{3m} \left(2^{m+2} - 4 - 3B\right) \ .$$

Notons respectivement A_2 et A_4 le nombre d'occurrences de 2 et de 4 dans la table des différences de F. On sait que

$$\sum_{a \in \mathbb{F}_2^m \setminus \{0\}} \sum_{b \in \mathbb{F}_{2^m}} \delta(a, b) = 2A_2 + 4A_4 = (2^m - 1)2^m \ ,$$

d'où nous dérivons que

$$\begin{aligned} \sum_{a,b \in \mathbb{F}_2^m} \delta(a, b)^2 &= 2^{2m} + 4A_2 + 16A_4 \\ &= 2^{2m} + 2^{m+1}(2^m - 1) + 8A_4 \ . \end{aligned}$$

Ainsi,

$$\begin{aligned} 8A_4 &= 2^m \left(2^{m+2} - 4 - 3B - 2^{m+1} - 2\right) \\ &= 2^m \left(2^{m+1} - 2 - 3B\right) \ , \end{aligned}$$

ce qui implique le résultat. Notablement, F est APN, *i.e.* $A_4 = 0$ si et seulement si

$$2^{m+1} - 2 - 3B = 0 \ ,$$

et $A_2 = 0$ si et seulement si $A_4 = 2^{m-2}(2^m - 1)$, *i.e.*, $B = 0$. □

Nous avons vu dans la proposition 2.8 que, lorsque $\alpha = \beta = 1$, la table des différences des Papillons correspondants ne contient pas de 2. Le dernier item de la proposition précédente montre que cette situation correspond au cas où $\mathsf{V}_{\alpha,\beta}$ n'a pas de composantes courbes. En d'autres mots, les coefficients de Walsh de $\mathsf{V}_{1,1}$ et de $\mathsf{H}_{1,1}$ sur $\mathbb{F}_{2^n} \times \mathbb{F}_{2^n}$ prennent les valeurs 0 et $\pm 2^{n+1}$ seulement. Ce résultat, correspondant au théorème 5 dans [LW14], est alors une conséquence directe de la proposition 2.8.

Plus généralement, le spectre de Walsh entier et la table des différences des Papillons généralisés peuvent être obtenus en appliquant la proposition précédente à $\mathsf{V}_{\alpha,\beta}$.

Corollaire 2.4 (Spectres de Walsh et différentiel des Papillons généralisés)**.** *Soient α et β deux éléments non-nuls de $\mathbb{F}_{2^n}$ tels que $\beta \neq (1+\alpha)^3$. Le spectre de Walsh de $\mathsf{H}_{\alpha,\beta}$ et $\mathsf{V}_{\alpha,\beta}$,* i.e.*, le multi-ensemble*

$$\{|\widehat{\mathsf{H}_{\alpha,\beta}}(u,v)|, u \in \mathbb{F}_2^{2n}, v \in \mathbb{F}_2^{2n}\backslash\{0\}\} = \{|\widehat{\mathsf{V}_{\alpha,\beta}}(u,v)|, u \in \mathbb{F}_2^{2n}, v \in \mathbb{F}_2^{2n}\backslash\{0\}\}$$

est donné par

$\widehat{\mathsf{H}_{\alpha,\beta}}(u,v)$	0	$\pm 2^n$	$\pm 2^{n+1}$
occurrences	$3 \times 2^{2n-2}(2^n-1)(2^n+1-C)$	$2^{2n}(2^n-1)C$	$2^{2n-2}(2^n-1)(2^n+1-C)$

où $(2^n-1)C$ est le nombre de composantes courbes de $\mathsf{V}_{\alpha,\beta}$.

Les tables des différences de $\mathsf{H}_{\alpha,\beta}$ et $\mathsf{V}_{\alpha,\beta}$ contiennent les valeurs 0, 2 et 4 avec les nombres d'occurrences suivants :

$\delta(a,b)$	2	4
occurrences	$2^{2n-2}(2^n-1) \times 3C$	$2^{2n-3}(2^n-1)(2^{n+2}+4-3C)$

Démonstration. Ce résultat se déduit directement de la proposition 2.9, en utilisant que le nombre de composantes courbes de $\mathsf{V}_{\alpha,\beta}$ est de la forme $B = (2^n-1)C$. En effet, en notant $f_{\lambda,\mu}$ pour λ, μ in $\mathbb{F}_{2^n}$, les composantes de $\mathsf{V}_{\alpha,\beta}$, *i.e.*,

$$f_{\lambda,\mu} : (x,y) \mapsto \mathrm{Tr}(\lambda R(x,y)) + \mathrm{Tr}(\mu R(y,x)) \ ,$$

nous avons par la preuve du théorème 2.22 que, pour tout $\mu \in \mathbb{F}_{2^n}^*$, $f_{0,\mu}$ n'est pas courbe et que, pour tout $\lambda \in \mathbb{F}_{2^n}$ non-nul, $f_{\lambda,\mu}$ est courbe si et seulement si $f_{1,\lambda^{-1}\mu}$ est courbe. Nous en déduisons que $B = (2^n-1)C$ où C est le nombre de $\mu \in \mathbb{F}_{2^n}$ tels que $f_{1,\mu}$ est courbe. Le spectre de Walsh des Papillons généralisés est alors obtenu en réalisant que tous les coefficients de Walsh d'une composante courbe sont égaux à $\pm 2^n$ et, pour une composante de linéarité 2^{n+1}, la transformée de Walsh prend 2^{2n-2} fois la valeur $\pm 2^{n+1}$ et $(2^{2n} - 2^{2n-2})$ fois la valeur 0 (cf. proposition 2.6).

Le spectre différentiel se déduit de la proposition 2.9 : le nombre de 4 dans la table des différences est

$$A_4 = 2^{m-2}(2^m-1) - 3 \times 2^{m-1}B = 2^{2n-3}(2^n-1)\left(2^{n+1}+2-3C\right) \ ,$$

et le nombre de 2 est

$$A_2 = (2^{2n}-1)2^{2n-1} - 2A_4 = 2^{2n-2}(2^n-1) \times 3C \ .$$

□

Les valeurs de C pour tout $\mathsf{H}_{\alpha,\beta}$ de $2n$ variables, avec $n \in \{3,5\}$ sont données aux tables 2.1 et 2.2. Nous avons vérifié par ordinateur pour $n \in \{3,5,7,9\}$ que, lorsque α et β varient dans $\mathbb{F}_{2^n}^*$, prend la valeur 0 et toutes les valeurs paires entre $(\frac{2^n+4}{3} - 2^{(n-1)/2})$

et $(\frac{2^n+4}{3} + 2^{(n-1)/2})$. Ceci implique que, pour ces valeurs de n, la famille des Papillons généralisés contient des applications avec $(2^{(n-1)/2} + 2)$ spectres de Walsh différents (et de ce fait le même nombre de tables des différences).

Nous soulignons également que la famille des Papillons généralisés inclue des applications qui ne sont pas équivalentes CCZ aux applications de la famille étudiée dans [PUB16]. En effet, le cas $\beta = 1$ n'inclut pas toutes les valeurs possibles de C.

Nous pouvons aussi vérifier à partir des valeurs de C que la plupart des Papillons généralisés ne pas équivalents CCZ aux permutations d'uniformité différentielle 4 connues jusqu'alors. Par exemple, la table 2.1 montre que les Papillons généralisés en 6 variables ont une table des différences avec un nombre de 4 égal à l'une des quatre valeurs suivantes :

$$A_4 \in \{0, 336, 672, 1008\} .$$

Si le dernier cas correspond au même spectre différentiel que les permutations puissances de Gold et Kasami, les fonctions avec les deux valeurs intermédiaires de A_4 ne peuvent pas être équivalentes à des fonctions puissances. En effet, les seules fonctions puissances d'uniformité différentielle 4 satisfont $A_4 = 1008$ (Gold et Kasami) ou $A_4 = 63$ (la fonction inverse). Plus généralement, pour tout nombre de variables, aucun des Papillons généralisés n'est équivalent CCZ à la fonction inverse puisque la formule de A_4 dans la preuve du corollaire 2.4 montre que A_4 est pair, alors que la fonction inverse a un nombre impair de 4 dans sa table des différences. Similairement, on peut vérifier que la seule permutation de $\mathbb{F}_2^6$ de non-linéarité optimale construite dans [Qu+13] a un autre spectre différentiel (voir [Qu+13, Table II]). Remarquons aussi que les Papillons généralisés à six variables ont une non-linéarité supérieure à celle des fonction construites dans [TCT15] et dans [ZHS14].

Table 2.1 – Valeur de C, *i.e.*, nombre de composantes courbes divisé par $(2^3 - 1)$, de tout $\mathsf{H}_{\alpha,\beta}$ pour α et β dans $\mathbb{F}_{2^3}^*$ où $\mathbb{F}_{2^3}$ est défini par un élément primitif a tel que $a^3 + a + 1 = 0$.

$\alpha\backslash\beta$	1	a	a^2	a^3	a^4	a^5	a^6
1	0	4	4	4	4	4	4
a	**6**	2	0	2	**6**	0	0
a^3	2	4	2	0	2	4	2

Table 2.2 – Valeur de C, *i.e.*, nombre de composantes courbes divisé par $(2^5 - 1)$, de tout $\mathsf{H}_{\alpha,\beta}$ pour α et β dans $\mathbb{F}_{2^5}^*$ où $\mathbb{F}_{2^5}$ est défini par l'élément primitif a tel que $a^5 + a^2 + 1 = 0$.

$\alpha\backslash\beta$	1	a	a^2	a^3	a^4	a^5	a^6	a^7	a^8	a^9	a^{10}	a^{11}	a^{12}	a^{13}	a^{14}	a^{14}	a^{16}	a^{17}	a^{18}	a^{19}	a^{20}	a^{21}	a^{22}	a^{23}	a^{24}	a^{25}	a^{26}	a^{27}	a^{28}	a^{29}	a^{30}
0	0	16	16	12	16	12	12	8	16	12	12	12	12	12	8	12	16	12	12	8	12	12	12	12	12	8	12	12	8	12	12
1	12	10	12	12	8	10	12	10	10	12	10	8	12	12	10	12	10	14	12	10	12	10	14	0	14	10	12	10	12	14	10
3	14	8	10	10	14	12	8	10	12	12	12	12	10	8	12	14	10	10	8	14	14	8	10	12	14	0	14	12	10	8	14
5	14	0	10	10	12	12	0	12	12	10	10	0	14	10	14	12	12	10	14	14	12	12	12	12	14	14	10	12	12	14	10
7	12	12	14	16	0	16	14	12	12	12	8	10	10	14	14	10	0	10	14	12	12	14	10	0	10	14	14	10	10	8	12
11	8	16	14	16	8	14	12	14	10	10	10	10	10	10	14	12	14	8	16	14	16	8	0	14	12	10	0	10	12	14	0
15	12	14	10	16	12	8	12	10	10	10	0	10	10	10	12	8	12	16	10	14	12	14	10	12	8	10	10	8	12	10	14

2.3.5.5 Du degré algébrique

Il est bien connu que, si les coefficients de Walsh d'une fonction booléenne f de m variables sont divisibles par 2^ℓ, alors le degré algébrique de f est au plus $(m + 1 - \ell)$ (vois e.g. [Lan90, proposition 1.5]). Ainsi, nous déduisons du théorème 2.22 que le degré algébrique de tout Papillon généralisé de $2n$ variables ne dépasse pas $(n+1)$. Nous montrons maintenant que cette borne supérieure est en fait atteinte pour presque toutes les valeurs de (α, β).

Théorème 2.25. *Soient α et β deux éléments non-nuls de $\mathbb{F}_{2^n}$. Le Papillon généralisé ouvert $\mathsf{H}_{\alpha,\beta}$ a un degré algébrique égal à n ou $n+1$. Il est égal à n si et seulement si*

$$(1+\alpha\beta+\alpha^4)^3 \;=\; \beta(\beta+\alpha+\alpha^3)^3.$$

Le Papillon fermé $\mathsf{V}_{\alpha,\beta}$ est de degré algébrique 2.

Remarque 2.5. La condition $(1+\alpha\beta+\alpha^4)^3 \;=\; \beta(\beta+\alpha+\alpha^3)^3$ peut être écrite de manière alternative $Z(\alpha,\beta)=0$, où :

$$Z(\alpha,\beta) \;=\; \beta^4+\alpha\beta^3+\alpha(\alpha+1)^6\beta+(1+\alpha)^{12}.$$

Qui plus est, Z peut se factoriser de la façon suivante :

$$\begin{aligned} Z(\alpha,\beta) &= \beta^4+\alpha\beta^3+\alpha(\alpha+1)^6\beta+(1+\alpha)^{12} \\ &= \left(\beta^2+(1+\alpha)^6\right)\left(\beta^2+\alpha\beta+(1+\alpha)^6\right) \\ &= \left(\beta^2+(1+\alpha)^6\right)\left(1+\alpha\beta+\alpha^4+(\beta+\alpha+\alpha^3)^2\right). \end{aligned}$$

Ainsi, si $\beta \neq (1+\alpha)^3$ alors $Z(\alpha,\beta)=0$ si et seulement si $1+\alpha\beta+\alpha^4=(\beta+\alpha+\alpha^3)^2$. Il s'ensuit que $Z(\alpha,\beta)$ est égal à 0 quand $\beta=(1+\alpha)^3$ et, si $\mathrm{Tr}(\alpha^{-1})=1$, pour les deux valeurs additionnelles de β. Ceci inclut le cas des réseaux de Feistel, lorsque $\alpha=\beta=1$.

Démonstration.
À l'évidence, $\mathsf{V}_\alpha(x,y)$ est de degré algébrique 2. Nous nous focalisons donc sur le Papillon généralisé ouvert $\mathsf{H}_{\alpha,\beta}$. La partie gauche de la sortie d'un tel Papillon ouvert est égale à $(x+\beta y^3)^{1/3}+\alpha y$, où (x,y) est l'entrée. Nous déduisons du théorème 1 de [KS12] (ou de manière équivalente de la proposition 5 de [Nyb94b]) que l'inverse de 3 modulo (2^n-1) pour n impair est

$$1/3 \;\equiv\; \sum_{i=0}^{(n-1)/2} 2^{2i} \mod (2^n-1),$$

ce qui implique en particulier que le degré algébrique de $x \mapsto x^{1/3}$ vaut $(n+1)/2$. Nous déduisons de cette expression que la fonction $t(x,y)=(x+\beta y^3)^{1/3}$ est égale à $\prod_{i=0}^{(n-1)/2}(x+\beta y^3)^{2^{2k}}$. Cette somme peut se développer en :

$$t(x,y)=(x+\beta y^3)^{1/3}=\sum_{J\subseteq[0,(n-1)/2]} \underbrace{\prod_{j\in J}\beta^{2^{2j}}y^{3\times 2^{2j}}}_{\deg<2|J|} \underbrace{\prod_{j\in\overline{J}}x^{2^{2j}}}_{\deg=(n+1)/2-|J|},$$

où $\overline{J}$ est le complément de J dans $[0,(n-1)/2]$, i.e. $J\cap\overline{J}=\varnothing$ et $J\cup\overline{J}=[0,(n-1)/2]$. Le degré algébrique de chaque terme de la somme est au plus égal à $|J|+(n+1)/2$. Ainsi, si $|J|<(n-1)/2$, le degré du terme correspondant est inférieur à n. Si $\overline{J}=\varnothing$, alors le terme correspondant est égal à $\beta^{1/3}y$ et est de degré 1. Si $\overline{J}=\{j\}$ pour un certain j, alors le terme est égal à

$$x^{2^{2j}}\times\beta^{1/3}y\times(\beta y^3)^{2^n-1-2^{2j}}=\beta^{1/3-2^{2j}}\times x^{2^{2j}}\times y^{(2^n-1)-(2^{2j+1}+2^{2j}-1)}.$$

Si $j\neq(n-1)/2$, alors son degré algébrique est

$$1+n-wt(2^{2j+1}+2^{2j}-1)=n-2j.$$

Si $j=(n-1)/2$, alors le terme (en omettant son facteur constant) vaut

$$x^{2^{n-1}}\times y\times y^{2^n-1-(2^n-2^{n-1})}=x^{2^{n-1}}y^{2^{n-1}-1}.$$

et est de degré n. De fait, $t(x,y)$ a deux termes de degré n, correspondant à $j=0$ et $j=(n-1)/2$ à savoir

$$m_0(x,y)=\beta^{-2/3}xy^{2^n-3} \text{ et } m_1(x,y)=\beta^{(2^{n-1}-1)/3}x^{2^{n-1}}y^{2^{n-1}-1}$$

Ainsi, la partie gauche de la sortie est de degré algébrique n. La partie droite vaut

$$L(x,y) \;=\; \Big(y+\alpha\big((x+\beta y^3)^{1/3}+\alpha y\big)\Big)^3+\beta\big((x+\beta y^3)^{1/3}+\alpha y\big)^3,$$

que nous pouvons réécrire en utilisant la fonction $t(x,y)=(x+\beta y^3)^{1/3}$:

$$L(x,y) \;=\; \big((\alpha^2+1)y+\alpha t(x,y)\big)^3+\beta\big(t(x,y)+\alpha y\big)^3,$$

que nous développons en

$$\begin{aligned}L(x,y) \;&= t(x,y)^3(\alpha^3+\beta) \;+\; y^3\big((\alpha^2+1)^3+\alpha^3\big)\\ &\quad+\; yt(x,y)^2\big((\alpha^2+1)\alpha^2+\beta\alpha\big) \;+\; y^2t(x,y)\big((\alpha^2+1)^2\alpha+\beta\alpha^2\big).\end{aligned}$$

Les termes de la première ligne sont de degré au plus 3. Focalisons-nous sur la deuxième ligne et notons la somme des termes $L'(x,y)$. Dans un premier temps, nous pouvons simplifier l'expression comme suit :

$$\frac{L'(x,y)}{\alpha} \;=\; C_0yt(x,y)^2 \;+\; C_1y^2t(x,y)$$

où $C_0=(\beta+\alpha+\alpha^3)$ et $C_1=(1+\alpha\beta+\alpha^4)$.

Comme $t(x,y)$ est de degré algébrique n, nous déduisons que $L'(x,y)$ (et la partie droite de la sortie de $\mathsf{H}_{\alpha,\beta}$) est de degré algébrique au plus $(n+1)$, alors que la fonction au complet est de degré au moins n à cause de la partie gauche. De plus, cette borne supérieure est atteinte si et seulement si les termes de degré $(n+1)$ dans $L'(x,y)$ ne s'annulent pas mutuellement. Les seuls termes dans $L'(x,y)$ qui peuvent être de degré $(n+1)$ correspondent aux termes de degré n dans $t(x,y)$, à savoir (en omettant les facteurs constants) :

$$\begin{aligned}y^2m_0(x,y)=xy^{2^n-1}, &\quad y^2m_1(x,y)=x^{2^{n-1}}y^{2^{n-1}+1},\\ ym_0(x,y)^2=x^2y^{(2^n-1)-3}, &\quad ym_1(x,y)^2=xy^{2^n-1}\;.\end{aligned}$$

Seuls les premier et dernier termes sont de degré $(n+1)$. Ainsi, le terme de degré $(n+1)$ dans $L'(x,y)$ est :

$$\begin{aligned}C_0ym_1(x,y)^2+C_1y^2m_0(x,y) \;&= xy^{2^n-1}\left(C_0\beta^{-1/3}+C_1\beta^{-2/3}\right)\\ &= xy^{2^n-1}\beta^{-1/3}\left(C_0+C_1\beta^{-1/3}\right)\;.\end{aligned}$$

Il s'ensuit que $\mathsf{H}_{\alpha,\beta}$ est de degré $(n+1)$ si et seulement si

$$\beta C_0^3\neq C_1^3\;.$$

□

2.3.6 Conclusion sur les Papillons généralisés

Avec le recul, ces travaux sur les Papillons généralisés nous apprennent beaucoup de choses. Premièrement, qu'il est possible de créer des boîtes-S sur 6 bits structurées, donc avec une implémentation peu coûteuse, et en même temps APN, donc optimales pour la résistance aux attaques différentielles.

Ensuite, que notre généralisation des Papillons (et les autres généralisations qui ont suivi) ne permettent pas de construire d'autres Papillons APN qu'un Papillon équivalent à la boîte-S de Dillon (sur 6 bits).

Ce résultat est certes décevant, mais ces généralisations permettent néanmoins de construire des boîtes-S structurées sur n'importe quel nombre de bits $2n$, n impair, avec une uniformité différentielle de 4 et une linéarité de 2^{n+1}, ce qui correspond à la meilleure résistance connue pour une boîte-S sur un nombre pair de bits (hormis la boîte-S de Dillon). Ces boîtes-S structurées sont moins coûteuses que d'implémenter directement une fonction de $2n$ bits.

Malheureusement, il y a tout de même des améliorations à trouver, car les tailles qui nous intéressent sont généralement les tailles de la forme 2^n, $n \geqslant 2$, et donc pas les tailles $2n$, n impair. Dans ce cas les Papillons offrent de moins bons résultats et les réseaux de Feistel sont pour l'instant plus adaptés.

De plus, diviser l'implémentation d'une fonction de $2n$ bits en l'implémentation de plusieurs fonctions de n bits est un bon progrès, mais pour n grand, cela reste coûteux. Il peut donc être intéressant de s'intéresser à décomposer des fonctions de mn bits en fonctions de n bits de manière plus générale [13], et les bonnes propriétés du Papillon dans le cas $m = 2$ est prometteur pour trouver des décompositions efficaces dans un cas plus général.

Une autre question de fond se pose dans les travaux sur les Papillons. On s'aperçoit qu'il est possible de « déplier » un Papillon ouvert de degré $n + 1$ en un Papillon fermé de degré 2 par CCZ équivalence. Une telle transformation simplifie énormément l'étude, et la question se pose de savoir s'il est possible d'appliquer des transformations similaires à d'autres constructions pour en simplifier l'étude.

Bien évidemment, le grand problème APN reste ouvert : on ne sait toujours pas s'il est possible de trouver d'autre permutation APN que la boîte-S de Dillon sur un nombre pair de bits. Les généralisations imaginées pour les Papillons n'ont pour l'instant pas permis de résoudre ce problème, mais il n'est pas impossible qu'une autre généralisation ne puisse amener à des permutations APN.

2.4 Aller plus loin : paramétrages, implémentations et réflexions

2.4.1 Implémentation bitsliced : réduire les coûts, optimiser le masquage

2.4.1.1 Objectifs et idée

Comme spécifié dans l'introduction (section 1.5.1), il est important d'obtenir des chiffrements peu coûteux à implémenter. Dans le cadre des chiffrements par blocs, cela passe par minimiser les coûts d'implémentation des deux composants majeurs d'un chiffrement par bloc : la fonction de diffusion et la boîte-S.

Nous cherchons ici à minimiser le coût d'implémentation d'une boîte-S, c'est-à-dire que nous cherchons une boîte-S peu coûteuse avec des bonnes propriétés de résistance à la cryptanalyse. L'approche usuelle [14] est de commencer par chercher une boîte-S avec de bonnes propriétés, puis de chercher une implémentation peu coûteuse de cette boîte-S (comme dans [Osv02 ; BP10] par exemple). L'inconvénient de cette approche est que trouver l'implémentation la moins coûteuse pour une boîte-S donnée est un problème Σ_2^P-complet (plus que NP-complet).

13. Comme le font par exemple les réseaux de Feistel généralisés.
14. Du moins c'était encore le cas à l'époque de la publication de ces travaux.

Nous préférons ici l'approche qui consiste à commencer par identifier les implémentations peu coûteuses, et chercher parmi elles une implémentation qui donne une boîte-S avec de bonnes propriétés, dans la lignée d'Ullrich *et al.* [Ull+11]. Nous construisons des descriptions de boîtes-S en circuits logiques, et nous testons leurs propriétés cryptographiques jusqu'à trouver un bon candidat.

Des analyses des réseaux de Feistel, des réseaux de type MISTY, nous pouvons déduire que les réseaux de Feistel sont particulièrement bien adaptés pour l'implémentation de boîtes-S (cf. table 2.4). Nous choisissons donc de construire des boîtes-S de 8 bits à partir de 3 tours d'un réseau de Feistel. Comme il est faisable d'explorer les implémentations de boîtes-S de 4 bits, nous générerons 3 fonctions de 4 bits S_1, S_2 et S_3 avec de faibles coûts d'implémentation, et testerons si le réseau de Feistel correspondant a de bonnes propriétés.

Dans le théorème 2.9, nous obtenons des conditions nécessaires sur S_1, S_2 et S_3 pour que le réseau de Feistel correspondant ait une uniformité différentielle optimale. En particulier, il faut que S_1 soit APN, que S_2 soit bijective avec $\delta(S_2) = 4$ et que S_3 soit APN.

Nous cherchons donc des circuits de faible coût pour implémenter des fonctions APN sur 4 bits et des permutations de 4 bits d'uniformité différentielle 4. En les insérant dans un réseau de Feistel F, avec bonne probabilité, on aura $\delta(F) = 8$ et $\mathcal{L}(F) = 64$.

2.4.1.2 Recherche des meilleurs candidats

Dans la lignée d'Ullrich *et al.*, nous effectuons une recherche orientée vers les implémentations bit-sliced. Nous considérons des séquences d'opérations logicielles, avec les instructions `and`, `OR`, `XOR`, `NOT`, et `MOV`, utilisant au plus 5 registres. Cela se traduit directement en implémentation matérielle : l'instruction `MOV` devient un branchement tandis que les autres instructions représentent les portes logiques correspondantes.

Notre algorithme de recherche génère une arborescence de circuits (séquences d'instructions). À chaque étape, nous considérons le nœud le moins coûteux de l'arborescence et générons tous ses descendants en ajoutant chacune des instructions possibles à la suite de son circuit. En passant, nous testons si les propriétés cryptographiques du nœud (en tant que fonction) sont les propriétés-cibles, et si oui nous arrêtons l'algorithme. Par construction, nous obtenons donc le circuit le moins coûteux qui implémente une fonction avec les propriétés recherchées.

Il y a 85 choix d'instructions possibles à chaque étape, mais nous pouvons utiliser des relations d'équivalence pour réduire l'espace de recherche.

Pour S_2, nous avons besoin d'une permutation de 4 bits d'uniformité différentielle 4. Cette recherche est peu coûteuse, et avait d'ailleurs déjà été réalisée par Ullrich *et al.* ([Ull+11],), nous n'avons eu qu'à reprendre leurs résultats.

Pour S_1 et S_3, nous avons implémenté une version de leur algorithme pour trouver des fonctions APN. Nous avons trouvé qu'il est impossible de construire une fonction APN sur 4 bits avec moins de 10 instructions. Il y a des solutions à 10 instructions, mais elles contiennent au minimum 6 instructions non-linéaires (`and`, `OR`), ce qui n'est pas efficace pour le masquage (cf. section 1.2.4.1). Enfin, avec 11 instructions, il existe des constructions de fonctions APN avec 4 instructions non-linéaires, 5 instructions `XOR` et 2 instructions `MOV` (copie).

Cette recherche a demandé de l'ordre de 6000 heures-cpu de calcul. Le facteur de branchement de notre recherche est proche de 10, alors qu'Ullrich *et al.* indiquent un facteur de branchement de moins de 7 ; ceci est dû au fait qu'on ne restreint pas la recherche aux seules permutations.

Grâce à cette recherche, nous obtenons des boîtes-S de 8 bits très efficaces avec de bonnes propriétés cryptographiques, utilisant 12 portes non-linéaires et 26 XORs (dans $\mathbb{F}_2^4$). Il en résulte des boîtes-S de 8 bits très efficaces avec de bonnes propriétés cryptographiques, ne coûtant que 12 portes non-linéaires et 26 XORs.

Le lemme suivant prouve que ce nombre de portes non-linéaires est optimal.

Lemme 2.14. *Soit S une permutation de 4 bits avec $\delta(S) \leqslant 4$ ou une fonction de 4 bits APN. Toute implémentation de S demande au moins 4 portes non-linéaires.*

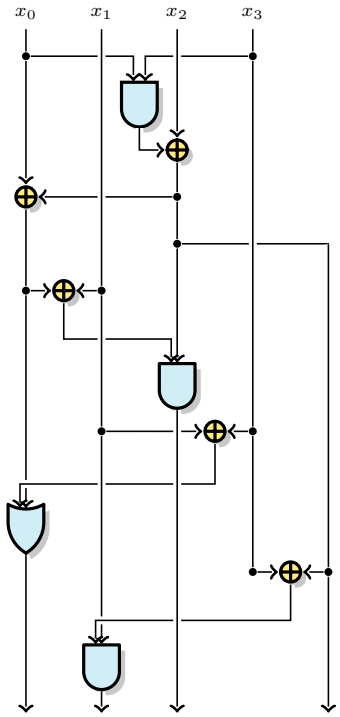
Démonstration. Si S peut être implémentée avec 3 portes non-linéaires ou moins, alors le degré algébrique de l'expression des variables de sortie est une combinaison linéaire des variables d'entrées et de 3 polynômes correspondant aux sorties des 3 portes non-linéaires. Ainsi, il existe une combinaison linéaire des variables d'entrée et sortie qui somme à une constante, *i.e.* $\mathcal{L}(S) = 16$.

D'après la classification des permutations de 4 bits dans [De 07], toute permutation avec $\delta(S) = 4$ satisfait $\mathcal{L}(S) \leqslant 12$. De plus, la classification des fonctions APN sur 4 bits [BL08] montre qu'elles satisfont $\mathcal{L}(S) = 8$, ce qui prouve le lemme. □

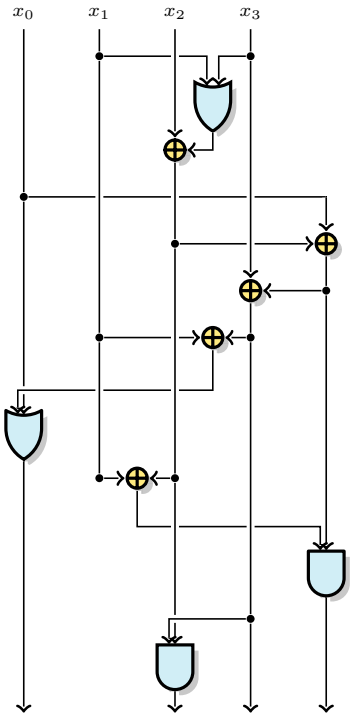
2.4.1.3 Quelques exemples de résultats

Les résultats exposés dans les figures suivantes (figures 2.15, 2.16 et 2.17) sont quelques exemples parmi beaucoup de boîtes-S bijectives sur 8 bits avec $\delta = 8$ et $\mathcal{L} = 64$, construites par 3 tours de Feistel à bas coût.

Dans les figures 2.15 et 2.16, la boîte-S APN sur 4 bits est la même pour S_1 et S_3. Ce n'est absolument pas un cas fréquent, mais cela permet plus de simplicité, d'où le choix de ces exemples. En particulier, ces boîtes-S de 8 bits sont involutives (puisque l'inverse d'un réseau de Feistel consiste à échanger l'ordre des boîtes-S internes).

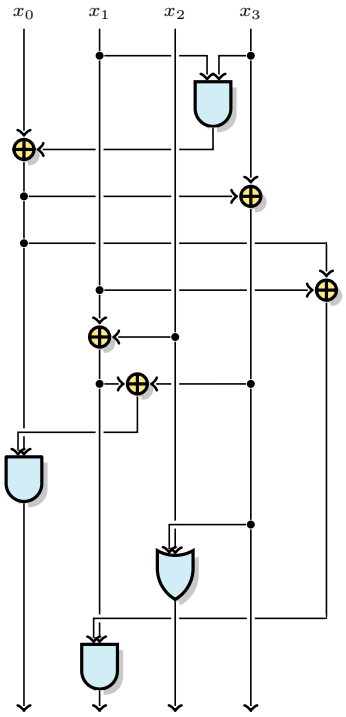


(a) Boîte-S APN sur 4 bits (S_1 et S_3) utilisant 3 and, 1 or et 5 xor.

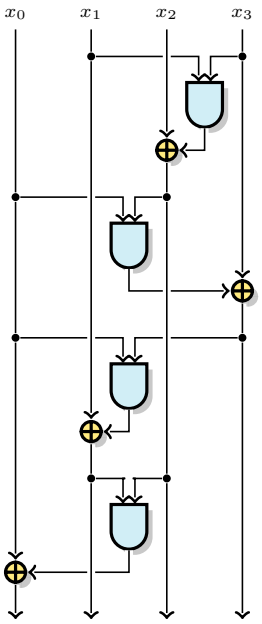


(b) Boîte-S bijective sur 4 bits (S_2) utilisant 2 and, 2 or et 4 xor.

Figure 2.15 – Un réseau de Feistel sur 3 tours utilisant le circuit de gauche pour S_1 et S_3 et le circuit de droite pour S_2 atteint une sécurité $\delta = 8$, $\mathcal{L} = 64$ en 8 and, 4 or, 14 xor ($+(3 \times 4)$ xor pour les 3 xor sur 4 bits d'un réseau de Feistel).



(a) Boîte-S APN sur 4 bits (S_1 et S_3) utilisant 3 and, 1 or et 5 xor.



(b) Boîte-S bijective sur 4 bits (S_2) utilisant 4 and, 0 or et 4 xor.

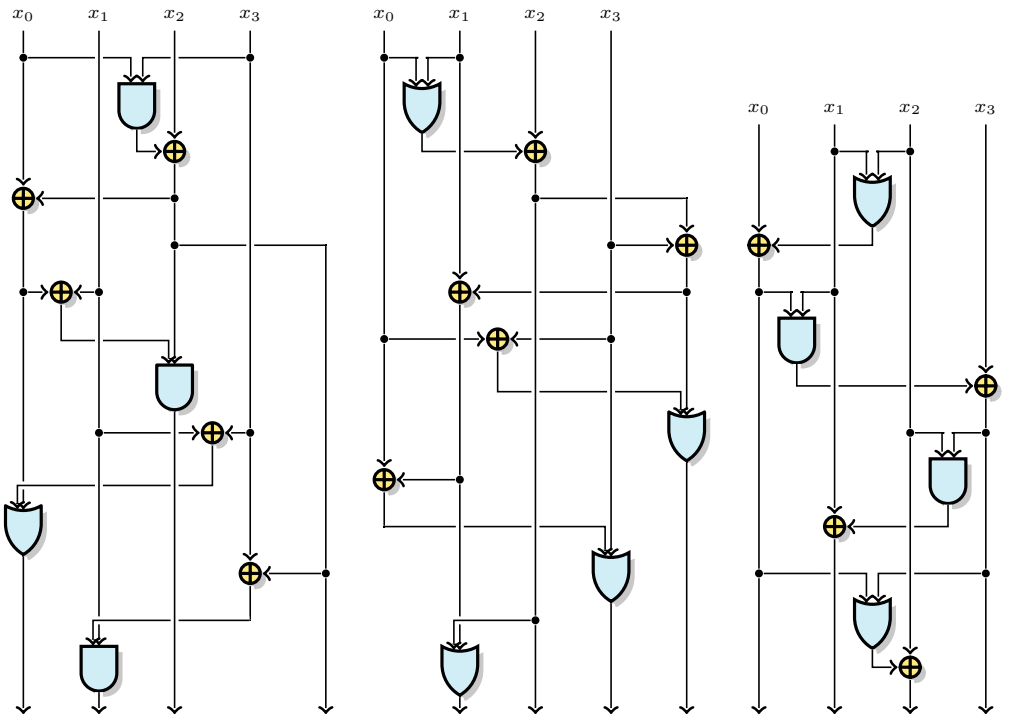
Figure 2.16 – Un réseau de Feistel sur 3 tours utilisant le circuit de gauche pour S_1 et S_3 et le circuit de droite pour S_2 atteint une sécurité $\delta = 8$, $\mathcal{L} = 64$ en 8 and, 4 or, 14 xor ($+(3 \times 4)$ xor pour les 3 xor sur 4 bits d'un réseau de Feistel).

2.4.1.4 Aparté sur les réseaux de type MISTY déséquilibrés

Une variante des réseaux de type MISTY trouve un intérêt particulier ici. Il s'agit des réseaux de type MISTY déséquilibrés, qui consistent à prendre la branche de droite et la branche de gauche de tailles différentes. Ainsi, on peut séparer les 8 bits en 3 à gauche et 5 à droite. Par conséquent, le réseau n'utilisera que des boîtes-S sur 3 et 5 bits. L'avantage de celles-ci est qu'on connaît des boîtes-S bijectives et APN sur n'importe quel nombre impair de bits, et donc en particulier sur 3 et 5 bits.

En utilisant cette variante, nous obtenons des boîtes-S de 8 bits S bijectives avec $\delta(S) = 8$, ce qui est mieux que le cas des MISTY équilibrés et aussi bien que le cas des réseaux de Feistel. La contrepartie est que cette variante nécessite d'implémenter des boîtes-S sur 5 bits, ce qui est plus compliqué que d'implémenter des boîtes-S sur 4 bits.

Exemple 2.2. Considérons un réseau de type MISTY déséquilibré sur 3 tours, avec deux permutations de 5 bits S_1 et S_3 et une permutation de 3 bits S_2. Après S_1 et S_3, la branche x_L sur 3 bits est xorée aux 3 bits de poids fort de x_R ; après S_2, les 3 bits de poids fort de la branche x_L (sur 5 bits) sont xorés dans x_L (cf. figure 2.18. Ce schéma donne une boîte-S



(a) Boîte-S APN sur 4 bits (S_1) utilisant 3 and, 1 or et 5 xor.

(b) Boîte-S APN sur 4 bits (S_3) utilisant 0 and, 4 or et 5 xor.

(c) Boîte-S bijective sur 4 bits (S_2) utilisant 2 and, 2 or et 4 xor.

Figure 2.17 – Un réseau de Feistel sur 3 tours utilisant les deux circuits de gauche pour S_1 et S_3 respectivement et le circuit de droite pour S_2 atteint une sécurité $\delta = 8$, $\mathcal{L} = 64$ en 8 and, 4 or, 14 xor ($+(3 \times 4)$ xor pour les 3 xor sur 4 bits d'un réseau de Feistel).

bijective de 8 bits avec $\delta = 8$, $\mathcal{L} = 64$ pour les choix de S_1, S_2 et S_3 suivants :

$$\begin{aligned} S_1 &= [\texttt{00}, \texttt{01}, \texttt{02}, \texttt{04}, \texttt{03}, \texttt{08}, \texttt{0d}, \texttt{10}, \texttt{05}, \texttt{11}, \texttt{1c}, \texttt{1b}, \texttt{1e}, \texttt{0e}, \texttt{18}, \texttt{0a}, \\ &\quad \texttt{06}, \texttt{13}, \texttt{0b}, \texttt{14}, \texttt{1f}, \texttt{1d}, \texttt{0c}, \texttt{15}, \texttt{12}, \texttt{1a}, \texttt{0f}, \texttt{19}, \texttt{07}, \texttt{16}, \texttt{17}, \texttt{09}] \\ S_2 &= [2, 5, 6, 4, 0, 1, 3, 7] \\ S_3 &= [\texttt{00}, \texttt{01}, \texttt{02}, \texttt{04}, \texttt{03}, \texttt{08}, \texttt{10}, \texttt{1c}, \texttt{05}, \texttt{0a}, \texttt{1a}, \texttt{12}, \texttt{11}, \texttt{14}, \texttt{1f}, \texttt{1d}, \\ &\quad \texttt{06}, \texttt{15}, \texttt{18}, \texttt{0c}, \texttt{16}, \texttt{0f}, \texttt{19}, \texttt{07}, \texttt{0e}, \texttt{13}, \texttt{0d}, \texttt{17}, \texttt{09}, \texttt{1e}, \texttt{1b}, \texttt{0b}] \end{aligned}$$

On peut donc noter que généraliser nos analyses des réseaux de Feistel et des réseaux de type MISTY au cas déséquilibré peut apporter des bons résultats, en particulier dans le cas des constructions de type MISTY.

En particulier, il est possible que des réseaux de Feistel ou de type MISTY déséquilibrés puissent atteindre $\delta = 4$ et $\mathcal{L} = 32$ sur 8 bits. C'est une question ouverte qui mériterait de s'y intéresser.

2.4.2 Comparaison avec les autres boîtes-S de 8 bits

Les boîtes-S de 8 bits que nous obtenons sont excellentes. Elles n'atteignent pas la sécurité de la boîte-S d'AES (dont on ne connaît pas d'équivalent), mais atteignent une sécurité au moins aussi bonne que toutes les alternatives à coût moindre.

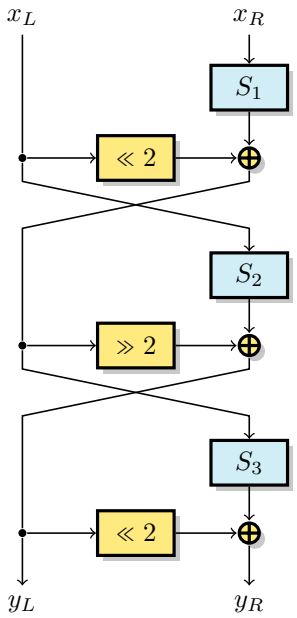


Figure 2.18 – Schéma d'un réseau de type MISTY déséquilibré. x_L est sur 5 bits, x_R sur 3. $\ll 2$ fait un décalage de 2 bits vers la gauche en introduisant 2 zéros aux bits de poids fort. $\gg 2$ fait un décalage de 2 bits vers la droite (les 2 bits de poids faible sont perdus).

En particulier, nous atteignons une meilleure uniformité différentielle que les boîtes-S utilisées dans Robin et Fantomas [Gro+15], avec un faible surcoût.

Pour comparer les valeurs respectives des boîtes-S considérées à la table 2.3, nous utilisons que, en première approximation, le nombre de tours nécessaires pour atteindre une sécurité fixée contre les attaques différentielles est proportionnel à $1/\log(\delta(S)/256)$, et que le coût d'implémentation par tour est proportionnel au nombre de portes non-linéaires (pour une implémentation logicielle bit-sliced utilisant du masquage). Ceci nous permet de dériver une mesure du coût d'implémentation simple pour les boîtes-S, donnée en dernière colonne, en prenant 1 comme coût pour celle de l'AES, et en considérant uniquement la sécurité face aux attaques différentielles.

Notons que nous ne considérons ici que des boîtes-S offrant une bonne résistance cryptographique. Dans la comparaison considérée (et dans la wide-trail-strategy), il est possible que des boîtes-S avec de moins bonnes propriétés cryptographiques et un coût très bas, itérées un grand nombre de fois, apportent de meilleurs résultats (une étude plus poussée de la façon de choisir les composants d'un chiffrement par blocs a été développée dans [Kho+14] par exemple).

À la suite de ces travaux, la boîte-S du chiffrement authentifié SCREAM ([Gro+14]) a été mise à jour. La boîte-S actuelle (SCREAM v3) est un réseau de Feistel de 8 bits avec uniformité différentielle 8 et linéarité 64, construite comme dans nos travaux. Les auteurs de SCREAM ont effectué une recherche supplémentaire pour obtenir une boîte-S qui évite les attaques par sous-espaces invariants.

Une autre table plus détaillée regroupant les mêmes boîtes-S que la table 2.3 et d'autres, focalisée sur les implémentations avec masquage (pour résister aux attaques par canaux auxiliaires) peut être trouvée dans [Bos+16], un article de Boss *et al.* qui pousse plus loin l'étude que nous avons amorcée sur la résistance aux canaux auxiliaires des boîtes-S de 8 bits. Leurs résultats ne sont pas cités ici car la comparaison n'est pas directe : pour un réseau de Feistel utilisant 3 tours identiques, Boss *et al.* permettent de n'implémenter qu'un seul tour et de l'itérer 3 fois, ce qui n'est pas considéré dans nos travaux.

Table 2.3 – Comparaison de quelques boîtes-S de 8 bits. La dernière colonne présente le coût d'implémentation global relatif (avec 1 pour AES).

		Implémentation		Propriétés		
S-Box	Construction	and/or	xor	$\mathcal{L}$	δ	Coût
AES [BP10]	Inversion dans $\mathbb{F}_{2^8}$+affine	32	83	32	4	1
Whirlpool [BR+00]	Lai-Massey	36	58	64	8	1.35
CRYPTON [Lim99]	Feistel 3 tours	49	12	64	8	1.83
Robin [Gro+15]	Feistel 3 tours	12	24	64	16	0.56
Littlun [Kar16]	Lai-Massey	12	24	64	16	0.56
Fantomas [Gro+15]	MISTY 3 tours (3/5 bits)	11	25	64	16	0.51
Whirlpool+Class13 [Gro+15]	Lai-Massey	16	41	64	10	0.64
Nôtres [CDL16]	Feistel 3 tours	12	26	64	8	0.45

D'autre part, ces travaux sur les réseaux de Feistel et les réseaux de type MISTY sont la source de l'implémentation Sage de fonctions sur les boites-S : sage.crypto.sbox [15] intègre ces constructions dans ses fonctionnalités (ces ajouts ont été effectués par Rusydi Makarim en 2016).

2.4.3 Boîtes-S à bas coût, une étude zoologique : discussion

2.4.3.1 Comparaison entre les Papillons et les réseaux de Feistel et de type MISTY

J'aimerais ici faire remarquer plusieurs différences majeures sur les résultats entre les réseaux de Feistel et de type MISTY d'une part, et les Papillons d'autre part.

Pour les notations, considérons des réseaux de Feistel et de type MISTY et des Papillons sur $2n$ bits.

- Les bornes sur les réseaux de Feistel et de type MISTY sont des bornes *inférieures* sur δ et $\mathcal{L}$: *aucun* réseau de Feistel ou de type MISTY n'offrira une meilleure sécurité que ces bornes.
- Les bornes sur les Papillons généralisés sont des bornes *supérieures* sur δ et $\mathcal{L}$: *tout* Papillon généralisé aura au moins une sécurité aussi bonne que ces bornes.
- Les bornes sur les Papillons ne valent que pour des choix très spécifiques des fonctions sur n bits, là où les résultats sur les réseaux de Feistel et de type MISTY sont génériques, *i.e.* indépendants du choix des fonctions de n bits. Étudier des cas plus particuliers pour les réseaux de Feistel et de type MISTY peut permettre de donner des bornes plus précises.
- D'ailleurs, le cas particulier du Papillon avec $\alpha = 1$ qui est un réseau de Feistel sur 3 tours avec des fonctions de n bits particulières permet d'obtenir des résultats plus précis que le cas générique. Notons que ces réseaux de Feistel sur 3 tours avec pour fonctions internes x^{2^i+1}, $x^{\frac{1}{2^i+1}}$ et x^{2^i+1} avaient été étudiés différemment par Li et Wang [LW14].

Pour résumer l'état de l'art après ma thèse sur la construction de boîtes-S, il semble que :

- pour $n = 3$, la façon la plus efficace de construire une boîte-S sur 6 bits est le Papillon,
- pour $n \neq 3$ impair, la façon la plus efficace de construire une boîte-S sur $2n$ bits est un réseau de Feistel sur 3 tours avec pour fonctions internes x^{2^i+1}, $x^{\frac{1}{2^i+1}}$ et x^{2^i+1} (ce qui correspond à un Papillon avec $\alpha = 1$) [16],

15. http://doc.sagemath.org/html/en/reference/cryptography/sage/crypto/sbox.html
16. En effet, un Papillon demande 4 fonctions internes, ce qui sera toujours plus coûteux qu'un réseau

— pour $n = 4$, la façon la plus efficace de construire une boîte-S sur 8 bits est un réseau de Feistel sur 3 tours,
— pour $n \neq 4$ pair, il n'y pour l'heure aucun résultat clair qui permette de conclure sur le meilleur choix de structure.

Table des résultats. La table 2.4 résume ces travaux sur les boîtes-S. *Toutes les bornes données sont atteintes*[17]. Le coût en nombre de fonctions internes correspond à une implémentation naïve de la structure[18]. Le coût optimisé en nombre de portes binaires correspond à optimiser l'implémentation de chacune des fonctions internes. Ce travail mériterait d'être effectué pour toute la table, mais il s'agit d'un travail complexe que je réserve pour des travaux futurs.

de Feistel qui en demande 3.

17. En revanche, les bornes génériques pour n quelconque ne sont pas nécessairement atteintes pour tout n.

18. Le cas du réseau de type MISTY déséquilibré nécessite 2 fonctions sur 5 bits et une sur 3 bits.

Table 2.4 – Comparaison des coûts d'implémentation et des bornes atteintes pour les réseaux de Feistel (3 tours), les réseaux de type MISTY (3 tours) et les Papillons.

Structure F	**Taille**	$\delta(F)$	$\mathcal{L}(F)$	**Coût**			**Informations**
				Fonctions internes	**Optimisé (portes)**		
					and/or	xor	
MISTY	$2n$	$\delta(F) \geqslant 4$	$\mathcal{L}(F) \geqslant 2^{n+1}$	3	-	-	-
Feistel	$2n$	$\delta(F) \geqslant 4$	$\mathcal{L}(F) \geqslant 2^{n+1}$	3	-	-	Bijectif
Papillon	$2n$, $n > 3$ impair	$\delta(F) = 4$	$\mathcal{L}(F) = 2^{n+1}$	4	-	-	Exposants 3×2^i, bijectif
Feistel	$2n$, n impair	$\delta(F) = 4$	$\mathcal{L}(F) = 2^{n+1}$	3	-	-	Papillon avec $\alpha = 1$, bijectif
MISTY	8	$\delta(F) \geqslant 16$	$\mathcal{L}(F) \geqslant 64$	3	12	24	Bijectif
MISTY	8	$\delta(F) \geqslant 8$	$\mathcal{L}(F) \geqslant 64$	3	12	26	Non-bijectif
MISTY	8	$\delta(F) \geqslant 8$	$\mathcal{L}(F) \geqslant 64$	3*	-	-	Déséquilibré (5/3), bijectif
Feistel	8	$\delta(F) \geqslant 8$	$\mathcal{L}(F) \geqslant 64$	192	12	26	Bijectif
Feistel	6	$\delta(F) \geqslant 4$	$\mathcal{L}(F) \geqslant 32$	72	-	-	Papillon avec $\alpha = 1$, bijectif
Papillon	6	$2 \leqslant \delta(F) \leqslant 4$	32	72	-	-	Bijectif

Chapitre 3

Diffusion : de la boîte-S au chiffrement par blocs

Les travaux précédents étudient la composante non-linéaire des chiffrements par blocs usuels : la boîte-S, qui permet d'apporter de la confusion. Dans les travaux de ce chapitre, nous nous intéresserons à la fonction qui assure la diffusion : la matrice de diffusion.

Il s'agit de travaux effectués avec Gaëtan Leurent et publiés à ToSC en 2018 [DL18b].

Vu de loin, les boîtes-S servent à rendre complexe la relation entre leur entrée et leur sortie, et la matrice de diffusion sert à rendre « actives » le plus de boîtes-S possible (la définition exacte de « active » venant de la wide-trail strategy).

L'objectif des travaux de ce chapitre est de construire des matrices de diffusion dont la diffusion est optimale avec le coût d'implémentation le plus bas possible. Les travaux sur le sujet sont innombrables, mais nous prenons une approche différente des travaux précédents qui nous permet d'obtenir des résultats drastiquement meilleurs que tous ceux d'avant 2018 et meilleurs que les résultats plus récents.

Les critères de sécurité souhaitables pour une matrice de diffusion sont de hauts facteurs de branchement et un faible coût d'implémentation (cf. wide-trail strategy, section 1.3.4).

Rappelons les définitions des facteurs de branchement :

Définition 3.1 (Facteur de branchement différentiel)**.** Soit M une matrice de dimension $m \times m$ sur des mots dans $\mathbb{F}_2^n$, *i.e.* $M \in M_k\big(M_n(\mathbb{F}_2)\big)$. On appelle facteur de branchement différentiel noté $\mathcal{B}_d(M)$ la valeur

$$\mathcal{B}_d(M) = \min_{x \neq 0}\{w(x) + w(M(x))\}\,.$$

Définition 3.2 (Facteur de branchement linéaire)**.** Soit M une matrice de dimension $m \times m$ sur des mots dans $\mathbb{F}_2^n$, *i.e.* $M \in M_k\big(M_n(\mathbb{F}_2)\big)$. On appelle facteur de branchement linéaire noté $\mathcal{B}_l(M)$ la valeur

$$\mathcal{B}_l(M) = \min_{x \neq 0}\{w(x) + w(M^\intercal(x))\}\,,$$

où $M^\intercal$ est la transposée de M.

Rappelons aussi les bornes connues sur ces facteurs de branchement :

$$\mathcal{B}_d(M) \leqslant m + 1$$

$$\mathcal{B}_l(M) \leqslant m + 1\,.$$

En particulier, on sait que si $\mathcal{B}_d(M) = m + 1$, alors $\mathcal{B}_l(M) = m + 1$ et réciproquement. On appelle matrice MDS une matrice telle que $\mathcal{B}_d(M) = \mathcal{B}_l(M) = m + 1$, et matrice presque-MDS une matrice telle que $\mathcal{B}_d(M) = \mathcal{B}_l(M) = m$.

Table 3.1 – Comparaison des matrices MDS les moins chères (A_4 est la matrice compagnon de X^4+X+1, A_8 est la matrice compagnon de $X^8+X^2+1=(X^4+X+1)^2$).

			Coût			
Taille	Anneau	Matrice	Naïve	Meilleure	Profondeur	Ref
$M_4(\mathbb{F}_2^8)$	$\mathbb{F}_{2^8}$	M_{AES}	152	97	3	[Kra+17]
	$\mathbb{F}_2[\alpha]$	M_{AES}	136	100	3	section 3.2
	$GL(8,\mathbb{F}_2)$	Circulante	106			[LW16]
	$GL(8,\mathbb{F}_2)$	Sous-corps		72	6	[Kra+17]
	$\mathbb{F}_2[\alpha]$	$M_{4,6}^{8,3}$	161	**67**	5	Fig. 3.8 avec $\alpha=A_8$
	$\mathbb{F}_2[\alpha]$	$M_{4,4}^{8,4''}$	160	**69**	4	Fig. 3.15 avec $\alpha=A_8$
	$\mathbb{F}_2[\alpha]$	$M_{4,3}^{9,5}$	145	**77**	3	Fig. 3.16 avec $\alpha=A_8$
$M_4(\mathbb{F}_2^4)$	$\mathbb{F}_{2^4}$	$M_{4,n,4}$	58	58	3	[Jea+17]
	$\mathbb{F}_{2^4}$	Toeplitz	58	58	3	[SS16]
	$\mathbb{F}_{2^4}$	Hadamard		36	6	[Kra+17]
	$\mathbb{F}_2[\alpha]$	$M_{4,6}^{8,3}$	87	**35**	5	Fig. 3.8 avec $\alpha=A_4$
	$\mathbb{F}_2[\alpha]$	$M_{4,4}^{8,4''}$	84	**37**	4	Fig. 3.15 avec $\alpha=A_4$
	$\mathbb{F}_2[\alpha]$	$M_{4,3}^{9,5}$	73	**41**	3	Fig. 3.16 avec $\alpha=A_4$

On sait construire des matrices MDS (par exemple à partir de codes MDS, Maximum Distance Separable). Un exemple notoire de matrice de diffusion MDS est la matrice MixColumns d'AES. Le problème est maintenant de trouver les matrices MDS les plus efficaces, c'est-à-dire les moins coûteuses.

3.1 La problématique des matrices de diffusion à bas coût

Dans cette section, j'introduis les différentes représentations des matrices de diffusion et comment on évalue leur sécurité en pratique, puis je passe en revue les travaux précédents sur le sujet en insistant plus sur les approches envisagées.

La table 3.1 résume les résultats obtenus dans nos travaux [DL18b].

Ces travaux combinent des idées provenant de plusieurs recherches. L'idée d'explorer les implémentations jusqu'à trouver une composante cryptographique appropriée a été appliquée notoirement aux S-Boxes par Ullrich *et al.* dans [Ull+11] et aux fonctions linéaires par exemple dans [Alb+14], alors que la classe de matrices que nous considérons est inspirée de travaux sur les matrices MDS récursives [Saj+12 ; WWW13 ; AF13].

Tout le code que nous utilisons pour nos recherches et pour vérifier nos résultats est disponible à l'adresse :
https://github.com/seduval/findmds.

3.1.1 Représentations et caractérisation des matrices MDS

Il existe diverses façons de représenter une matrice de diffusion. La représentation générique est comme une matrice à coefficients binaires, mais on peut aussi la représenter comme une matrice à coefficients dans un anneau voire un corps binaire.

Une autre manière de représenter une matrice de diffusion est par un circuit logique qui implémente la matrice vue comme fonction linéaire. Cette approche offre un avantage majeur pour l'étude des coûts, puisqu'on peut lire directement sur le circuit le coût global de l'implémentation de la matrice.

D'autre part, il est essentiel de pouvoir caractériser les matrices que l'on recherche, à savoir les matrices MDS, et ce de la manière la plus efficace possible, car tester si une matrice est MDS coûte cher même simplement pour des matrices 16×16.

3.1.1.1 Représentations

Soit M une matrice de dimension $N = m \times n$.

Représentation en matrice binaire. De manière générale, M bits peuvent être représentés par une matrice de dimension $N \times N$ à coefficients binaires, de sorte qu'un vecteur binaire de taille N pourra être multiplié à la matrice pour obtenir son image. Il s'agit donc alors d'un élément de $GL(N, \mathbb{F}_2)$.

L'avantage de cette représentation est qu'elle est très générique : il suffit d'avoir une structure d'espace vectoriel sur $\mathbb{F}_2$.

On peut représenter une matrice de diffusion de dimension $N = m \times n$ par une matrice de dimension $m \times m$ dont les coefficients sont sur n bits (un élément de $M_m(\mathbb{F}_2^n)$). Pour simplifier l'analyse ou les calculs, on peut demander que ces coefficients sur n bits suivent une certaine structure. Il s'agit d'un compromis : plus il y aura de structure, plus l'analyse sera simple, mais plus l'espace de recherche sera restreint (ce qui diminue le choix possible de matrices de diffusion). C'est pourquoi il existe une déclinaison de représentations sur des ensembles plus ou moins structurés.

Représentation en matrice sur un groupe. Une première structure peu contraignante est celle de groupe : on peut simplement considérer que chaque coefficient de n bits est lui-même une matrice de $GL(n, \mathbb{F}_2)$.

Représentation en matrice formelle sur un anneau. Si l'on impose que les coefficients appartiennent à un anneau commutatif unitaire, en particulier l'anneau des polynômes formels en α, $\mathbb{F}_2[\alpha]$, la matrice de diffusion M sera alors un élément de $GL(m, \mathbb{F}_2[\alpha])$. Il conviendra par la suite d'instancier l'indéterminée α par une fonction sur n bits.

Représentation en matrice sur un corps. En rajoutant un peu plus de structure, on peut considérer les coefficients de M comme des éléments du corps fini $\mathbb{F}_{2^n}$ (de la forme $GF(2)[X]/P(X)$, pour un choix de polynôme irréductible de degré n $P(X)$, de racine α). M sera alors un élément de $GL(m, \mathbb{F}_{2^n})$.

Il s'agit d'un choix usuel (notamment dans AES) puisque la structure de corps permet plus de facilité dans l'analyse.

Notation en entiers. Pour simplifier les notations, on représentera parfois les coefficients des matrices à coefficients dans un corps ou un anneau par des nombres entiers, où 1 sera l'élément neutre et 2 sera α (pour les différentes définitions de α dans le corps et dans les polynômes formels en α).

Représentation en circuit. En tant que circuit, une matrice $m \times m$ à coefficients de taille n pourra être représentée par un circuit logique prenant en entrée n registres et en sortant n (il peut y avoir plus de n registres en cours de calcul). Bien qu'il ne soit pas strictement impossible de calculer une fonction linéaire avec un circuit utilisant des opérations non-linéaires, nous considérerons ici par simplicité des circuits n'utilisant que des opérations linéaires (au niveau binaire, uniquement des xors et des copies de bits).

De façon équivalente, on peut représenter la matrice par un programme qui l'implémente. On utilise généralement la représentation en circuit pour les implémentations matérielles et la représentation en programme pour les implémentations logicielles.

3.1.1.2 Caractérisation

Il est bien connu que, sur un corps, on peut tester la propriété MDS de manière efficace : une matrice est MDS si et seulement si tous ses mineurs (les déterminants de ses sous-matrices carrées) sont non-nuls.

De même sur un anneau commutatif unitaire, une matrice est MDS si et seulement si ses mineurs sont non-nuls (différents du polynôme nul).

Notons que le coût de calcul du test de la propriété MDS est donc dans le pire des cas égale au calcul du déterminant de la matrice M. En effet, il s'agit de calculer l'expansion de Laplace du déterminant de M, qui décompose le déterminant de M est m déterminants de sous-matrices $(m-1) \times (m-1)$, et de considérer les expansions de Laplace de ces sous-matrices. Ainsi, calculer le déterminant de M revient à calculer les déterminants de toutes ses sous-matrices carrées, *i.e.* ses mineurs [1].

3.1.2 Matrices MDS à bas coût : une revue de l'existant

La recherche de matrices de diffusion à bas coût est prolifique. Généralement, on part du principe qu'on implémente la matrice comme une matrice carrée $k \times k$ à coefficients dans $\mathbb{F}_{2^n}$, et qu'appliquer la matrice revient à faire une multiplication matricielle, qui requiert $k \times (k-1)$ XORs dans $\mathbb{F}_{2^n}$ en plus des multiplications par chacun des coefficients de la matrice. De là, réduire le coût d'implémentation revient simplement à réduire le coût de la multiplication des coefficients, puisque le nombre de XORs $(k \times (k-1))$ est fixe. Il s'agit ainsi d'optimisation locale de chacun des coefficients. On cherchera à obtenir des matrices MDS avec des coefficients faciles à multiplier dans le corps (comme `1`, `2` ou `4`).

3.1.2.1 Optimisation locale

L'approche usuelle consiste à explorer toutes matrices dans un espace de recherche avec des coefficients faciles à implémenter (correspondant à des multiplications faciles), et à tester si les matrices sont MDS. L'astuce consiste alors à restreindre intelligemment l'espace de recherche en conservant des matrices MDS dans l'espace de recherche. Plusieurs espaces de recherche ont ainsi été considérés.

Matrices MDS récursives. Une idée importante pour réduire l'impact de l'implémentation des matrices MDS a été introduit par Guo, Peyrin et Poschmann dans la fonction de hachage à bas coût `PHOTON`[GPP11], et utilisée plus tard pour le chiffrement à bas coût `LED`[Guo+11]. L'approche consiste à considérer les matrices MDS M qui peuvent s'écrire sous la forme $M = A^j$, pour une matrice A peu chère à implémenter et un certain entier j. Ceci permet d'échanger de la rapidité d'implémentation contre de la taille d'implémentation : plutôt que d'implémenter directement M, on n'implémente que la matrice A, moins coûteuse, et on l'itère j fois.

Cette idée a été revisitée pour réduire plus avant le coût d'implémentation de A. En particulier, une série de travaux [Saj+12 ; WWW13 ; AF13] introduisent la notion de matrice MDS formelle, où les coefficients sont écrits comme des expressions abstraites en une fonction linéaire indéfinie α. Ceci permet de dériver un ensemble de conditions sur α pour que la matrice soit MDS, et de choisir un α avec un faible coût d'implémentation

1. Bien évidemment, dans un cas moyen, le test est plus court que le calcul de déterminant puisqu'on sait que la matrice n'est pas MDS dès qu'on trouve un mineur nul.

(typiquement un seulxor bit à bit). En particulier, ceci généralise la notion MDS des matrices sur un corps aux applications linéaires en général.

Optimiser les coefficients. Dans le contexte où la matrice MDS complète est vouée à être implémentée, plusieurs études s'intéressent à des classes particulières de matrices, telles que les matrices circulantes, Hadamard ou Toeplitz [Sim+15 ; LS16 ; SS16], en utilisant des coefficients qui peuvent être calculés efficacement. En particulier, certains de ces résultats considèrent des matrices involutives, c'est-à-dire égales à leur inverse.

De plus, l'idée de se détacher des opérations dans le corps fini pour considérer des opérations linéaires plus générales a aussi été appliquée dans [BKL16 ; LW16] et mène aux matrices MDS dans $M_4(M_8(\mathbb{F}_2))$ les moins chères jusqu'alors rapportées, avec un coût de 106 xors binaires [LW16]. En particulier, les techniques de Li et Wang peuvent être utilisées quand les coefficients de la matrice ne commutent pas.

3.1.2.2 Optimisation globale

Recherche d'implémentations à bas coût. Dans la construction de `PRIDE` [Alb+14], les auteurs utilisent une recherche sur les petites implémentations matérielles en utilisant des opérations sur les bits pour trouver une matrice sur 16 bits peu coûteuse, mais de facteur de branchement 4 sur des mots dans $\mathbb{F}_{2^4}$ (non-MDS).

Optimisation de l'implémentation avec des outils automatiques. Une autre approche est d'utiliser des outils pour chercher automatiquement des implémentations à bas coût de fonctions linéaires. Ce genre d'outils a été utilisé pour la première fois pour l'implémentation de fonctions cryptographiques dans [BMP13], où les auteurs utilisent des *straight line programs* (« programmes en ligne droite ») linéaires pour optimiser globalement l'implémentation d'une fonction linéaire prédéfinie. Dans ce papier, les auteurs prouvent que trouver l'implémentation optimale pour une fonction linéaire donnée est un problème NP-difficile, et ils développent des heuristiques pour optimiser une implémentation en utilisant des opérations linéaires au niveau des bits et en autorisant l'*annulation* (« cancellation ») de variables dans $\mathbb{F}_2$.

Il y avait déjà eu des premières tentatives pour utiliser des outils de synthèse pour optimiser des matrices MDS existantes dans $M_4(\mathbb{F}_2^8)$ (en particulier la matrice du MixColumns d'AES [Sat+01 ; Zha+16]), mais un grand pas a été franchi récemment par Kranz, Leander, Stoffelen et Wiemer [Kra+17]. Ils ont appliqué des outils d'optimisation de straight line programs linéaires au MixColumns d'AES, et à un grand nombre d'autres matrices MDS connues, et ont obtenu des implémentations significativement moins coûteuses. En particulier, ils ont réduit le coût du MixColumns d'AES de 103 à 97 xors bit à bit, et ont trouvé une matrice MDS qui peut être implémentée avec seulement 72 xors bit à bit alors que la meilleure était précédemment à 103 xors bit à bit.

Notre approche. Bien qu'il y ait déjà eu de nombreuses études de ce sujet [Sim+15 ; BKL16 ; LS16 ; LW16 ; SS16], la plupart d'entre elles se focalisent sur les coefficients de la matrice, en cherchant des matrices MDS dont beaucoup de coefficients sont faciles à évaluer (comme 1 ou 2). La supposition sous-jacente est que, pour toute ligne de la matrice, un circuit évaluera tous les coefficients et les additionnera, ce qui sous-entend un coût minimal de $k \times (k-1)$ XORs sur des mots pour une matrice $k \times k$. Notons qu'à l'origine, cette métrique de coût focalisé sur les coefficients a été imaginée pour des matrices séries. Pour les matrices séries, cette métrique permet effectivement d'étudier le coût avec précision, mais les études plus récentes s'étant éloignées des matrices séries, il devient plus intéressant d'évaluer le coût avec un compte de XORs du circuit complet.

C'est ce qu'on suggéré récemment Kranz *et al.* [Kra+17], qui ont appliqué des outils d'optimisation existants à des classes de matrices MDS de la littérature, et l'optimisation

globale effectuée par les outils a donné un gain significatif par rapport aux optimisations locales qui étaient utilisées jusqu'alors. En particulier, ces circuits sont bien plus petits que les $k \times (k-1)$ XORs sur des mots qui étaient considérés comme un minimum par les travaux précédents.

Dans notre travail, effectué en parallèle de [Kra+17], nous prenons une approche différente pour trouver des matrices MDS avec une implémentation optimisée globalement. Plutôt que d'optimiser une matrice donnée, nous lançons une recherche dans un ensemble de circuits, ordonnés par coût matériel, jusqu'à trouver un circuit qui corresponde à une matrice MDS. Le circuit peut réutiliser des valeurs intermédiaires, ce qui fait que l'optimisation globale réduit le nombre de portes requises. Comme le circuit pour un étage linéaire complet de 32 bits est assez grand, nous considérons une classe de circuits qui peuvent être représentés au niveau des mots, en utilisant des XORs sur les mots et des applications linéaires fixés. Le coût de calcul de cette exploration est tout de même important (en particulier en termes de mémoire utilisée), mais avec des optimisations, nous pouvons atteindre des matrices MDS de tailles 3×3 et 4×4 pour n'importe quelle taille de mots. Par construction, ces matrices sont optimales dans la classe de matrices considérée, et elles apportent une amélioration significative par rapport aux résultats précédents et nous obtenons des résultats meilleurs que ceux de [Kra+17], comme le montre la table 3.1.

Nos travaux combinent des idées de divers axes de recherche. L'idée d'explorer les implémentations jusqu'à trouver un composant cryptographique adapté a été notablement appliqué aux boîtes-S par *et al.* dans [Ull+11] et aux fonctions linéaires par exemple dans [Alb+14], là où la classe de matrices que nous considérons est inspirée de travaux précédents sur les matrices MDS récursives [Saj+12 ; WWW13 ; AF13].

D'une certaine manière, notre approche peut être interprétée comme la recherche de bons straight line programs, en revanche nous limitons le nombre de variables disponibles simultanément et nous n'utilisons que des opérations sur les mots, non sur les bits (alternativement, on pourrait dire que nous trouvons des straight line programs sur un anneau, en utilisant des additions dans l'anneau et des multiplications par des constantes). Nous notons que nos straight line programs utilisent des annulations.

Une autre particularité de notre approche est que, contrairement aux travaux précédents sur la recherche de petites implémentations comme [Alb+14], nous nous concentrons sur des opérations au niveau des mots, non des bits, et ce sans fixer la taille des mots.

Tout le code que nous avons utilisé pour la recherche et pour vérifier les résultats est disponible à l'adresse suivante :
https://github.com/seduval/findmds.

3.1.2.3 Choix de la métrique

Afin d'évaluer le coût matériel d'une opération linéaire, il nous faut compter le nombre de xors binaires utilisés dans l'implémentation (une implémentation n'étant à l'évidence pas unique). En général, une implémentation peut être décrite comme une séquence d'opérations $x_i \leftarrow x_{a_i} \oplus x_{b_i}$, avec $a_i, b_i < i$, où $x_1, \ldots, x_{n \times k}$ est l'entrée, et la sortie est un sous-ensemble des x_i. Ceci correspond à un straight line program linéaire. L'idéal serait de calculer le coût minimal d'une implémentation, mais ceci n'est pas toujours faisable en pratique, c'est pourquoi les métriques plus pratiques comptent les xors dans une classe d'implémentations plus restreinte.

Comptage de xors direct. Un décompte de xors direct a été proposé par Sim, Khoo, Oggier et Peyrin dans [Sim+15]. Ceci correspond à compter le nombre de portes utilisées dans une implémentation naïve d'un application linéaire. En considérant la matrice binaire qui représente le application linéaire dans $M_{nk}(\mathbb{F}_2)$, chaque ligne donne une formule pour calculer un bit de sortie, et s'il y a t bits non-nuls dans une ligne, cette formule peut être

Table 3.2 – Comparaison des métriques. Les résultats de type « +Yosys » ont été optimisés avec l'outil de synthèse Yosys (qui utilise ABC en sous-routine).

Matrice		Compte de xors				
$\mathbb{F}_4$	$\mathbb{F}_2$	Naïve	Naïve+Yosys	LIGHTER	Nôtres	Nôtres+Yosys
$\begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix}$	$\left[\begin{array}{cc\|cc\|cc} 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ \hline 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ \hline 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \end{array}\right]$	15	12	**10**		
$\begin{bmatrix} 3 & 2 & 2 \\ 2 & 3 & 2 \\ 2 & 2 & 3 \end{bmatrix}$	$\left[\begin{array}{cc\|cc\|cc} 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ \hline 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ \hline 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \end{array}\right]$	21	14	11	11	**10**

calculée en $t - 1$ portes. Ainsi, le décompte de xors direct est défini comme le nombre de bits à 1 dans la matrice binaire, moins $k \times n$.

Cette métrique a été énormément utilisée dans les travaux sur les matrices MDS à bas coût, tels que [Sim+15 ; LS16 ; LW16 ; SS16]. Par définition, pour ce choix de métrique, le coût d'une matrice MDS est égal au coût de l'évaluation de chacun de ses coefficients plus le coût de $k \times (k-1)$ XORs de mots de n bits.

Comptage de xors séquentiel. Le décompte de xors séquentiel défini dans [Jea+17] est une meilleure approximation du coût optimal d'implémentation, et a été utilisé pour optimiser les multiplications dans le corps utilisées dans des matrices MDS [BKL16 ; Jea+17]. Il s'agit de compter le nombre de xors bit à bit dans un programme séquentiel restreint aux opérations en-place [2] sans registres additionnels [3]. Cette mesure peut être significativement inférieure au décompte de xors direct, mais la restriction aux opérations en-place est tout de même très contraignante.

Dans le contexte des matrices MDS, cette métrique a été utilisée pour optimiser les coûts de multiplications dans le corps, mais en raison de coûts de calculs excessifs, elle n'a pas été utilisée pour optimiser des matrices MDS entières.

Optimisation globale. Plus récemment, des outils heuristiques de recherche de straight line programs linéaires ont été appliqués pour obtenir de bonnes implémentations d'une matrice MDS donnée [Kra+17]. Ceci donne des résultats bien meilleurs que toutes les implémentations précédentes qui n'exploitaient que des optimisations locales.

Dans nos travaux, nous considérons une classe d'implémentations légèrement restreinte. Nous décomposons l'évaluation d'une matrice MDS en une séquence d'étapes simples : des xors mot-à-mot et de simples opérations linéaires qui généralisent la multiplication dans un corps. Nous utilisons en plus des registres supplémentaires afin de permettre la réutilisation de valeurs intermédiaires. Dans cette classe d'implémentations, nous effectuons une recherche exhaustive en ciblant une matrice MDS. Comme nous le verrons, cette classe contient des implémentations de matrices MDS bien moins coûteuses que celles proposées précédemment.

2. Sans créer de valeurs intermédiaires.

3. Lorsque nous mentionnons des registres, nous utilisons le terme au sens logiciel : un registre est utilisé pour enregistrer des variables intermédiaires dans les calculs. Dans nos implémentations matérielles, ces registres sont remplacés par de simples fils.

Comparaison des métriques. Pour comparer les métriques, nous considérons deux matrices MDS dans $M_3(\mathbb{F}_4)$ à la table 3.2. La première matrice est optimale pour le comptage de xors direct et pour n'importe quelle métrique qui considère les coefficients indépendamment, avec un coût de $3+4\times 3$, là où la deuxième matrice est l'une de celles découvertes par notre outil, $M_{3,4}^{5,1}$, qui peut être implémentée efficacement comme montré en figure 3.4. Pour toute matrice, nous évaluons le décompte de xors direct (correspondant à l'implémentation naïve), le décompte de xors séquentiel donné par l'outil `LIGHTER` [Jea+17] (puisque l'on travaille sur des petites tailles, nous pouvons calculer le comptage de xors séquentiel de la matrice entière, plutôt que seulement celui des multiplications dans le corps), et le coût de l'implémentation naïve optimisée avec les outils de synthèse Yosys [4] et ABC [5]. Pour la deuxième matrice, nous évaluons l'implémentation trouvée par notre outil, et nous tentons de l'optimiser plus avant avec les outils Yosys et ABC.

Nous pouvons noter plusieurs résultats importants dans cette table. Tout d'abord, optimiser globalement plutôt que localement a un énorme impact, puisque cela réduit le décompte de xors de 21 ou 15 à 10. En particulier, les meilleures implémentations que nous ayons trouvées demandent moins de xors binaires que les 6 XORs sur des mots de 2 bits qui sont assimilés à un coût fixé dans la plupart des travaux précédents. Nous remarquons aussi que l'utilisation de registres additionnels peut être utile : la deuxième matrice a un coût optimal de 11 sans registres additionnels, mais il y a une implémentation avec seulement 10 xors en utilisant des registres additionnels. Enfin, soulignons que nos nouvelles constructions sont similaires aux matrices MDS précédentes en comparant à cette petite échelle, mais l'avantage de notre approche est qu'elle s'étend à des matrices de taille 4 sur des mots de 8 bits, alors que `LIGHTER` ne peut optimiser que des fonctions linéaires avec au plus 8 entrées.

Limitations. Malheureusement, compter le nombre de portesxor dans un circuit n'est pas nécessairement une bonne estimation du véritable coût matériel d'une implémentation, pour plusieurs raisons. En premier lieu, les outils de design matériel cherchent à optimiser le circuit. En particulier, les métriques considérées sont une surestimation du nombre minimal de xors binaires, et le coût relatif entre deux circuits peut changer si des optimisations supplémentaires sont trouvées (un exemple concret est l'implémentation naïve de la matrice MDS d'AES, qui requiert 152 xors binaires, alors que les outils de synthèse Yosys et ABC peuvent réduire ce coût à 115 xors binaires). Deuxièmement, les circuits matériels peuvent utiliser des portes autres que des xors à deux entrées. En particulier, les FPGA modernes possèdent des LUT relativement grandes (look-up tables, « table de recherche »), ce qui fait que des portesxor à plusieurs entrées ne sont pas beaucoup plus coûteuses que des portes à seulement deux entrées. À nouveau cela peut changer le coût relatif entre deux circuits, mais nous escomptons que cet effet soit limité dans le cas des synthèses ASIC (où une portexor à trois entrées est presque deux fois plus grande qu'une portexor à deux entrées). Finalement, un autre critère important à considérer est la profondeur du circuit, *i.e.* le nombre maximal de portes dans n'importe quel chemin entre entrée et sortie. Celle-ci impacte le délai de propagation du signal, qui définit la fréquence maximale à laquelle le circuit peut tourner, et impacte fortement sur les performances (en particulier, sur le débit par espace). Ceci sera géré dans nos travaux en proposant plusieurs matrices MDS atteignant divers compromis entre le décompte de xors et la profondeur de l'implémentation.

En général, nos constructions offrent un gain significatif par rapport aux propositions précédentes, et nous nous attendons à des gains réels dans des implémentations concrètes, malgré les limitations discutées ci-dessus. Nous utiliserons pour métrique le nombre de portesxor dans la suite, car cette métrique a été beaucoup utilisée précédemment et qu'il est difficile d'obtenir une meilleure estimation générique. Nous laissons une comparaison plus

4. http ://www.clifford.at/yosys/
5. https ://people.eecs.berkeley.edu/ alanmi/abc/

précise avec de véritables implémentations matérielles de matrices variées à des travaux futurs.

3.1.3 Notations

Nous noterons la taille d'une matrice MDS k, et la taille des mots n (*e.g.* la matrice MixColumns de l'AES correspond à $k = 4$ et $n = 8$). Nous utiliserons « XOR » pour signifier une addition de mots de n bits, et « xor binaire » (ou « xor bit à bit ») pour signifier une simple porte xor. En particulier, l'implémentation d'une opération XOR demande n xors binaires.

Plutôt que de considérer des matrices MDS sur un corps (*i.e.* chaque coefficient est une multiplication par un élément du corps), nous considérons une classe de matrices plus générale où chaque coefficient correspond à une opération linéaire dans $M_n(\mathbb{F}_2)$. Pour des raisons techniques, et suivant des travaux précédents [Saj+12 ; WWW13 ; AF13], nous restreignons les coefficients à des puissances d'une seule opération linéaire (en particulier, ceci assure que les coefficients commutent). Ainsi, les coefficients peuvent être écrits comme des polynômes dans $\mathbb{F}_2[\alpha]$, où l'inconnue α représente une opération linéaire indéfinie.

Notre recherche de matrices MDS a deux étapes : nous cherchons d'abord une matrices MDS formelle M avec des coefficients dans $\mathbb{F}_2[\alpha]$ (comme expliqué en section 3.3), puis nous sélectionnons un application linéaire A tel que $M(A)$ (la matrice où α est remplacée par A) soit MDS (comme expliqué en section 3.5).

En particulier, si α est instanciée par une multiplication F par un générateur d'un corps, $\mathbb{F}_2[F]$ est isomorphe au corps correspondant. Pour une notation compacte, nous représentons un polynôme par un entier avec la même représentation binaire ; par exemple, 2 représente l'élément α et 3 représente $\alpha + 1$.

3.2 Du MixColumns d'AES

Une matrice MDS importante est celle utilisée par l'opération MixColumns dans Rijndael, standardisé en tant qu'AES. Cette matrice est définie par :

$$M_{\text{AES}} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix},$$

où 1, 2 et 3 représentent les éléments dans le corps fini $\mathbb{F}_{2^8}$. Plus précisément, le corps fini est construit comme $\mathbb{F}_2[\alpha]/(\alpha^8 + \alpha^4 + \alpha^3 + \alpha + 1)$, et 2 et 3 sont les éléments α et $\alpha + 1$ respectivement.

Une implémentation naïve de cette matrice demande 1 multiplication par 2, 1 multiplication par 3 et 3 XORs pour chaque ligne. En matériel, une implémentation de ces opérations requiert respectivement 3 et 11 xors binaires, ce qui donne un coût total de $4 \times (3 + 11 + 3 \times 8) = 152$ xors binaires.

Pourtant, la meilleure implémentation connue de la multiplication par 3 dans le corps d'AES ne demande que 9 xors binaires [Jea+17], ce qui entraîne un coût total de 144 xors binaires. Alternativement, Zhao, Wu et Zhang ont utilisé une approche heuristique pour trouver une bonne séquence de xors binaires pour évaluer la matrice MDS d'AES (vu comme une matrice booléenne 32×32). Ils ont trouvé une représentation avec seulement 132 xors binaires dans [Zha+16].

En fait, nous pouvons obtenir de meilleurs résultats simplement en considérant les sous-expressions communes dans les calculs. En effet, une évaluation de M_{AES} peut s'écrire :

$$M_{\text{AES}} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 2a \oplus 3b \oplus c \oplus d \\ a \oplus 2b \oplus 3c \oplus d \\ a \oplus b \oplus 2c \oplus 3d \\ 3a \oplus b \oplus c \oplus 2d \end{bmatrix} = \begin{bmatrix} 2a \oplus 2b \oplus b \oplus c \oplus d \\ a \oplus 2b \oplus 2c \oplus c \oplus d \\ a \oplus b \oplus 2c \oplus 2d \oplus d \\ 2a \oplus a \oplus b \oplus c \oplus 2d \end{bmatrix}.$$

Avec cette expression, l'évaluation de M_{AES} ne demande que 4 multiplications par 2 (les valeurs $2a$, $2b$, $2c$, $2d$ dont utilisées deux fois) et 16 XORs ; ceci se traduit en 140 xors binaires, ce qui est moins coûteux que l'implémentation naïve. De plus, des valeurs intermédiaires peuvent être réutilisées. En particulier, chacune des valeurs $a \oplus b$, $b \oplus c$, $c \oplus d$ et $d \oplus a$ est utilisée deux fois si nous réécrivons la sortie :

$$\begin{bmatrix} 2a \oplus 3b \oplus c \oplus d \\ a \oplus 2b \oplus 3c \oplus d \\ a \oplus b \oplus 2c \oplus 3d \\ 3a \oplus b \oplus c \oplus 2d \end{bmatrix} = \begin{bmatrix} 2(a \oplus b) \oplus b \oplus (c \oplus d) \\ 2(b \oplus c) \oplus c \oplus (d \oplus a) \\ 2(c \oplus d) \oplus d \oplus (a \oplus b) \\ 2(d \oplus a) \oplus a \oplus (b \oplus c) \end{bmatrix}.$$

Avec cette formule, on peut évaluer la matrice avec seulement 12 XORs et 4 multiplications par 2, ce qui donne un coût total de seulement 108 xors binaires. À notre connaissance, cette astuce a été décrite pour la première fois en 2001 [Sat+01] et est utilisée dans l'implémentation Atomic-AES [BBR16].

Pour référence, la meilleure matrice 4×4 sur des mots de 8 bits obtenue à ce jour sans optimisation globale demande 106 xors binaire [LW16]. Ceci montre qu'optimiser l'implémentation d'une matrice MDS peut avoir un effet aussi important sur le coût final que le choix de la matrice MDS.

Choix du corps ou de l'anneau. Le choix du corps – ou de l'anneau – joue aussi un rôle important dans le coût d'implémentation d'une matrice MDS. Les matrices MDS sont typiquement définies sur des corps finis (comme mentionné plus tôt, le MixColumns d'AES est défini sur $\mathbb{F}_{2^8}$), mais ce choix n'est pas nécessaire et de meilleurs résultats peuvent être atteints sur des anneaux commutatifs.

En particulier, la construction par sous-corps utilisée dans [Ben+09 ; Bar+10 ; Kho+14] correspond à utiliser un anneau produit. Elle peut être appliquée à M_{AES} de la manière suivante : les entrées sont considérées comme des éléments de l'anneau $\mathbb{F}_{2^4} \times \mathbb{F}_{2^4}$, et les coefficients 1 dans la matrice sont interprétés comme $(1, 1)$, 2 comme (α, α) et 3 comme $(\alpha \oplus 1, \alpha \oplus 1)$, avec α un générateur du corps $\mathbb{F}_{2^4}$. Ceci correspond en fait à appliquer deux copies de M_{AES} défini sur $\mathbb{F}_{2^4}$, indépendamment sur chaque quartet de l'entrée. C'est intéressant car la multiplication par α dans $\mathbb{F}_{2^4}$ ne requiert qu'un seulxor binaire (car il existe des trinômes irréductibles de degré 4 sur $\mathbb{F}_2[X]$), alors que la multiplication par 2 dans $\mathbb{F}_{2^8}$ requiert trois xors binaires (car il n'y a pas de trinômes irréductibles de degré 8 dans $\mathbb{F}_2[X]$). Ainsi la multiplication par 2 dans l'anneau requiert seulement 2 xors binaires plutôt que 3 dans $\mathbb{F}_{2^8}$.

Plus généralement, nous pouvons considérer la matrice M_{AES} en tant que matrice formelle, où 1 représente l'identité, 2 une opération linéaire arbitraire α, et 3 l'opération linéaire $x \mapsto \alpha(x) \oplus x$ (en utilisant des idées et du formalisme de [Saj+12 ; WWW13 ; AF13 ; BKL16]). Les coefficients de la matrice sont maintenant des polynômes en α, *i.e.* des éléments de $\mathbb{F}_2[\alpha]$. Lorsque l'on instancie la matrice avec une transformation α donnée, la matrice sera MDS si et seulement si tous les mineurs sont inversibles. Les mineurs peuvent être aisément évalués en tant que polynômes ; dans ce cas, ils sont : $1, \alpha, \alpha \oplus 1, \alpha^2, \alpha^2 \oplus 1, \alpha^2 \oplus \alpha \oplus 1, \alpha^3 \oplus 1, \alpha^3 \oplus \alpha \oplus 1, \alpha^3 \oplus \alpha^2 \oplus 1, \alpha^3 \oplus \alpha^2 \oplus \alpha$.

En particulier, si tous les facteurs irréductibles du polynôme minimal de α sont de degré au moins 4, alors tous les mineurs seront inversibles. Concrètement, si α est la multiplication par un élément primitif d'un corps fini $\mathbb{F}_{2^n}$, le polynôme minimal de α est

irréductible et de degré n, donc la matrice sera MDS tant que $n \geqslant 4$. Dans l'AES, α n'est pas un élément primitif, mais son polynôme minimal est tout de même irréductible et de degré 8.

Nous pouvons dorénavant utiliser des opérations linéaires encore plus efficaces. Par exemple, l'opération $\alpha : x \mapsto (x \lll 1) \oplus ((x \gg 1) \wedge 1)$ peut s'implémenter très efficacement en matériel, avec simplement du fil et un seul xor binaire. Utilisée dans M_{AES}, elle génère aussi une matrice MDS, car son polynôme minimal est $(x^4 \oplus x \oplus 1)^2$. Cette nouvelle matrice MDS peut être implémentée avec seulement 100 xors binaires en utilisant l'astuce précédente.

Surprenamment, cette simple construction basée sur M_{AES} a en fait un coût d'implémentation moindre que la matrice MDS la moins coûteuse connue sur $M_4\big(M_8(\mathbb{F}_2)\big)$ (hormis celles obtenues par des optimisations globales dans [Kra+17]), et les mêmes idées donnent une matrice sur $M_4\big(M_4(\mathbb{F}_2^4)\big)$ avec un coût d'implémentation moindre que les constructions précédemment connues. Cet exemple motive l'approche que nous prenons dans ces travaux. Nous considérerons des matrices MDS définies sur un anneau, et nous étudierons un circuit qui implémente la matrice, plutôt que de seulement compter les 1 dans la matrice binaire. Nous chercherons de nouvelles matrices avec une implémentation efficace, en utilisant une représentation en tant que série d'opérations dans un corps (ou un anneau), et nous essaierons de minimiser le nombre de xors et d'opérations linéaires α requis pour obtenir une matrice MDS.

3.3 Algorithme de recherche par graphe

Une approche possible pour trouver des implémentations efficaces est d'effectuer une recherche exhaustive sur une classe d'implémentations et de tester si chaque implémentation correspond à la fonction cible. En particulier, un certain nombre de travaux précédents utilisent une approche par graphe pour faire cette recherche [Ull+11 ; Jea+17]. Une implémentation est représentée par une séquence d'opérations, ce qui définit implicitement un graphe où les nœuds sont des séquences d'opérations. Pour être plus précis, il y a une arête $L_1 \xrightarrow{\text{op}} L_2$ entre les séquences d'opérations L_1 et L_2 lorsque $L_2 = L_1, \text{op}$. Qui plus est, les séquences d'opérations qui définissent la même fonction à réordonnancement près des entrées et des sorties seront considérées comme équivalentes.

Trouver une implémentation optimale d'une fonction donnée (dans la classe d'implémentations correspondant aux opérations utilisées) revient alors à trouver le plus court chemin entre le circuit vide (correspondant à l'identité) et l'objectif. Alternativement, cette approche peut être exploitée pour trouver un membre optimal d'une classe de fonctions avec une certaine propriété prédéfinie.

3.3.1 Travaux précédents

Recherche par graphe. Cette approche a été utilisée pour la première fois pour créer des composants cryptographiques par Ullrich *et al.* [Ull+11], dans le contexte des boîtes-S bijectives de 4 bits. Ils travaillent sur 5 registres binaires, et génèrent un arbre d'implémentations dans lequel chaque transition consiste à ajouter une opération au circuit parmi l'ensemble de and, or, XOR, not et copy. L'opération not ne prend qu'un paramètre en entrée, les autres prennent deux paramètres (par exemple, l'opération and implémente la porte binaire associée : $x \leftarrow x \wedge y$), et il y a ainsi 85 choix possibles d'opération à chaque étape. Le registre supplémentaire peut stocker des valeurs intermédiaires, ce qui fait qu'utiliser des opérations non-inversibles ne donne pas nécessairement une boîte-S non-inversible. Plutôt que de chercher des implémentations d'une boîte-S fixée, ils cherchent

la meilleure implémentation pour toute boîte-S dans une classe d'équivalence donnée (à équivalence affine près).

Ils utilisent plusieurs règles afin de réduire le facteur de branchement ; en particulier, ils vérifient qu'il existe toujours un sous-ensemble de registres qui encode une permutation, et détectent lorsque deux nœuds sont équivalents. Ces règles réduisent beaucoup le facteur de branchement, et ils parviennent à trouver des implémentations en 9 instructions de permutations de 4 bits avec les meilleures uniformité différentielle et linéarité possible.

La métrique qu'ils utilisent pour comparer leurs implémentations est un simple décompte d'opérations, ce qui leur permet de réaliser un parcours en profondeur pour trouver le plus court chemin entre l'identité et la classe de boîtes-S.

Notons que, au chapitre 2, nous nous sommes déjà inspirés de cette approche pour obtenir des implémentations à bas coût de fonctions de 4 bits non-bijectives (en minimisant en priorité les portes non-linéaires, and et or) (cf. section 2.4.1.2,[CDL16]).

Recherche bidirectionnelle. Récemment, Jean, Peyrin, Sim et Tourteaux [Jea+17] ont utilisé une variante de cet algorithme avec une recherche bidirectionnelle. Ils se concentrent sur l'optimisation de l'implémentation d'une fonction donnée, et génèrent un arbre dans le sens direct en partant de l'identité, et dans le sens inverse en partant de la fonction cible. Ils utilisent aussi différents poids pour les instructions, ce qui fait de leur algorithme une variante bidirectionnelle de l'algorithme de Dijkstra, en utilisant une queue à priorité pour générer chaque arbre.

Ceci permet une recherche plus efficace que le parcours en profondeur d'Ullrich *et al.*, mais limite aux opérations inversibles. En particulier, ils ne peuvent pas utiliser de registre supplémentaire, et doivent combiner des portes élémentaires en opérations inversibles (comme par exemple $x \leftarrow x \oplus (y \wedge z)$). À cause de ces restrictions, la classe d'implémentations considérée est plus petite, et leurs implémentations sont potentiellement moins bonnes.

3.3.2 Idée générale de l'algorithme

Espace de recherche. Dans nos travaux, puisque nous cherchons des matrices MDS, nous utilisons r registres qui représentent des mots de $\mathbb{F}_2^n$ (plutôt que des bits), et nous considérons uniquement des opérations linéaires :

- XOR de deux mots ($x \leftarrow x \oplus y$) ;
- Copie d'un registre ($x \leftarrow y$) ;
- Application d'un application linéaire abstrait α à un registre $x \leftarrow \alpha(x)$, qui généralise la multiplication par un générateur α d'un corps fini.

Nous pouvons représenter les applications linéaires correspondant à une séquence d'instructions avec une matrice $k \times r$ à coefficients dans $M_n(\mathbb{F}_2)$. L'implémentation vide correspond à la matrice identité avec des colonnes à zéro additionnelles, et chaque opération d'une implémentation peut être traduite en une opération correspondante sur les colonnes de la matrice.

Test de MDS. Puisque $\mathbb{F}_2[\alpha]$ est un anneau commutatif, nous pouvons tester si une matrice est MDS en calculant ses mineurs[6] et en testant s'ils sont égaux au polynôme nul. Si un mineur est nul, alors tout choix de α donnera un mineur nul, et donc l'application linéaire correspondant a un facteur de branchement plus petit que k. En revanche, si tous les mineurs sont non-nuls, alors *certains* choix de α donneront un application linéaire de facteur de branchement maximal, lorsque n est assez grand[7] (voir la section 3.5 pour plus de détail).

6. Les déterminants de ses sous-matrices carrées.
7. Notons que si n est trop petit, il n'existe pas de matrice MDS de taille k.

Utilisation d'un algorithme de type A^*. Nous avons décidé d'utiliser des registres additionnels afin de permettre une meilleure implémentation. Ceci nous empêche d'utiliser une recherche bidirectionnelle, mais les propriétés fortes de l'ensemble des matrices MDS nous permettent d'utiliser un algorithme A^* afin de guider la recherche vers les matrices MDS.

L'algorithme A^* [HNR68] est une extension de l'algorithme de recherche de chemin de Dijkstra [Dij59]. Cette extension se spécialise dans la recherche de chemin entre une source unique et une destination, et utilise une estimation heuristique h de la distance restante entre le nœud et la destination. A^* explore itérativement le nœud qui minimise $g(x) + h(x)$, où $g(x)$ est la distance entre la source et x.

L'heuristique doit être admissible, *i.e.* elle ne doit jamais surestimer la distance restante, sinon l'algorithme peut trouver un chemin sous-optimal. De plus, l'heuristique est dite monotone si $h(x) \leqslant h(y) + d$ pour toute paire de nœuds x, y avec une arête de poids d. Lorsque l'heuristique est monotone, les nœuds n'ont besoin d'être explorés qu'un seule fois.

Heuristique. Dans notre cas, il nous faut une heuristique pour estimer le nombre d'opérations restantes avant d'atteindre une matrice MDS. Puisque les opérations affectent les colonnes de la matrice (et qu'une opération n'affecte qu'une seule colonne), nous comptons combien de colonnes de l'état courant pourraient faire partie d'une matrice MDS. Clairement, chaque colonne contenant des coefficients à zéro ne peut pas faire partie d'une matrice MDS. De plus, les colonnes qui sont linéairement dépendantes ne peuvent pas faire partie ensemble d'une matrice MDS. De ce fait, nous notons m le rang de la sous-matrice composée de toutes les colonnes sans coefficient à zéro. Notre heuristique considère qu'il faut encore au moins $k - m$ opérations XOR pour atteindre une matrice $k \times k$ MDS.

Il est simple de se rendre compte que cette heuristique ne surestime jamais le nombre d'opérations restantes, mais nous n'avons pas pu prouver qu'elle est monotone. Par contre, notre code teste la condition de monotonie à chaque évaluation de nœud, et nous n'avons jamais rencontré de situation qui viole cette condition [8].

L'utilisation d'A^* avec cette heuristique améliore significativement les performances de notre recherche, comparé à l'algorithme plus simple de Dijkstra.

3.3.3 Les grandes lignes de l'implémentation

Vu de loin, notre algorithme étend un arbre massif de fonctions sur lesquels nous testons la propriété MDS. Nous commençons avec la fonction identité, et à chaque nœud évalué, nous testons si la matrice correspondante est MDS, et nous générons un fils pour chaque opération possible dans notre ensemble.

Nous gardons en mémoire tous les nœuds créés dans deux structures : la première structure (`ÉtatsTestés`) contient tous les nœuds qui ont déjà été évalués, tandis que la seconde (`ÉtatsNonTestés`) stocke tous les nœuds qui ont été créés mais pas encore testés. À chaque étape de l'algorithme, nous sélectionnons un élément de poids estimé minimal dans `ÉtatsNonTestés`, et nous testons s'il est déjà dans `ÉtatsTestés`.

D'après A^*, le poids estimé d'un nœud est défini comme la somme du poids des opérations depuis la source, plus une estimation des opérations restantes pour atteindre une matrice MDS.

Notons que plusieurs chemins dans l'arbre peuvent amener au même état. C'est pourquoi, lorsque nous choisissons un nœud, nous testons d'abord s'il appartient déjà à `ÉtatsTestés`. Comme nous ouvrons les nœuds par ordre de poids estimé et que l'heuristique pour le poids restant vérifie (expérimentalement) la condition de monotonie, la première fois que nous évaluons un nœud correspond à une implémentation optimale.

8. Si cela devait arriver, nous aurions uniquement à évaluer le nœud une deuxième fois.

3.3.3.1 Réduction de l'espace de recherche

Afin de réduire le temps et la mémoire consommés par notre recherche, nous avons développé des optimisations pour réduire le branchement pendant la construction de l'arbre.

Tout d'abord, nous notons que permuter les entrées ou les sorties d'un circuit n'affecte ni son coût, ni la propriété MDS. De ce fait, nous considérons que les matrices sont équivalentes à réordonnancement près des entrées et sorties. En pratique, nous associons à chaque matrice un identifiant unique à réordonnancement près des mots d'entrée/sortie : nous considérons toutes les permutations des lignes et des colonnes, et nous utilisons pour identifiant la plus grande matrice (pour un certain ordre). En particulier, la structure `ÉtatsTestés` est remplacée par un ensemble d'identifiants `IDsTestés`. À chaque fois que nous tirons un nœud dans `ÉtatsNonTestés`, nous calculons son identifiant, et nous testons s'il est déjà présent dans `IDsTestés`.

Nous limitons aussi l'utilisation des opérations de copie, et nous considérons après une copie uniquement les circuits dont l'opération suivante est soit un application linéaire soit un XOR qui écrase la valeur copiée. Ceci ne limite pas l'ensemble des états atteignables (les circuits peuvent tous être réécrits pour respecter cette restriction), mais elle limite le branchement à chaque étape. Additionnellement, après une opération de copie, nous testons si le circuit est toujours injectif, et nous arrêtons l'exploration s'il ne l'est pas. En effet, une matrice MDS est nécessairement injective, et toutes les étapes intermédiaires doivent aussi être injectives.

Enfin, nous mettons des limites au coût et à la profondeur des circuits considérés, afin de ne pas générer des nœuds trop coûteux. Lorsque nous cherchons des matrices MDS de coût minimal, nous répétons l'algorithme en augmentant itérativement les limites, jusqu'à ce que la recherche réussisse avec le coût possible minimal.

L'algorithme résultant est donné en Algorithme 1.

Choix des coûts des opérations. Notre algorithme permet de travailler avec des coûts variables pour chaque opération, et bien choisir ces paramètres de coût est essentiel. Nous avons considéré le coût d'une copie à 0 (en matériel, il ne s'agit que de fil), et le coût de α à 1 (pour unxor binaire par défaut). Le coût d'un XOR mot-à-mot a été laissé variable, mais nous l'avons généralement considéré de coût 8 (*i.e.* 8 fois plus coûteux que α, ce qui correspond par exemple au cas de $\mathbb{F}_2^8$ avec α une multiplication par un élément primitif) ou un coût du XOR à 2 (coût minimal pour rester supérieur à α, pour des raisons de performances).

Pour ce qui est de la profondeur, nous considérons dans l'algorithme que les XOR et les opérations α sont de profondeur 1.

3.3.4 Extensions

Par-dessus l'algorithme principal, nous avons ajouté quelques extensions. Il s'agit d'extensions de l'ensemble des opérations, ce qui élargit la classe des matrices que nous considérons. Bien évidemment, une plus grande classe veut dire plus de matrices à tester, et donc nous pouvons trouver de nouvelles (et parfois meilleures) matrices, mais l'algorithme demande aussi plus de ressources. Ces extensions peuvent être utilisées séparément, mais des combinaisons sont possibles (en gardant à l'esprit que les extensions coûtent cher en performances ; en particulier, il n'est pas question d'utiliser toutes les extensions en même temps). Les premières extensions ajoutent des registres supplémentaires, tandis que les autres mettent à disposition plusieurs applications linéaires plutôt qu'un seul.

Algorithm 1 Algorithme pour chercher des circuits MDS.

```
1: function TROUVER MDS
   Entrée : POIDS_MAX, PROFONDEUR_MAX,poids des opérations.
2: Sortie : Toutes les matrices MDS de poids inférieur ou égal à POIDS_MAX.

3:     IDsTestés ← NULL
4:     ÉtatsNonTestés ← Identité
5:     PoidsCourant ← 0
6:     for état ∈ ÉtatsNonTestés avec état.poids = PoidsCourant do
7:         if TestedIDs.contient(state.ID) then
8:             continue
9:         if état.estMDS() then
10:            état.affiche)
11:            continue                ▷ Les fils sont équivalents ou de plus grand poids.
12:        état.génèreEnfants(ÉtatsNonTestés)
13:        if {ÉtatsNonTestés avec PoidsCourant} = ∅ then
14:            PoidsCourant ← PoidsCourant + 1
       return

15: function ÉTAT.GÉNÈREENFANTS(ÉtatsNonTestés)
16:    for op ∈ ensembleOp do
17:        étatFils ← état.ajouteOp(op)
18:        if étatFils.poids > POIDS_MAX ou étatFils.profondeur >
   PROFONDEUR_MAX then
19:            continue
20:        if op = copy et étatFils.nonInjectif() then
21:            continue
22:        ÉtatsNonTestés.ajoute(étatFils)
       return

23: function ÉTAT.AFFICHE Affiche l'état comme une matrice, donne son poids et ses
    opérations.

24: function ÉTAT.ESTMDS Teste si la fonction est MDS en calculant les déterminants de
    ses sous-matrices carrées.

25: function ÉTAT.NONINJECTIF Teste si la fonction est injective en calculant son déter-
    minant (il y a des subtilités car certains mots sont éliminés à la fin).

26: function ÉTAT.AJOUTEOP(op, origine, destination) Renvoie l'état fils à partir de
    l'état père et d'une nouvelle opération. Calcule le poids de l'enfant.
```

3.3.4.1 Registres additionnels en lecture (RO_IN).

La première extension ajoute des registres d'entrée supplémentaires pour stocker les k mots d'entrée (k nouveaux registres pour une matrice de taille k). Afin de limiter le facteur de branchement, ces registres sont en lecture seulement : ils peuvent être utilisés en entrée d'une opération XOR ou d'une copie, mais ne sont jamais modifiés. En particulier, avec cette extension, l'état complet est toujours injectif.

3.3.4.2 Utilisation de α^{-1} (INV).

Lorsque nous instancions α avec un application linéaire concret, nous choisissons généralement la matrice compagnon d'un polynôme creux (ou de manière équivalente, un LFSR) pour son faible coût en xors. Ceci implique que α^{-1} a aussi un coût en xors faible [BKL16], ce qui rend l'utilisation de α^{-1} intéressante. C'est pourquoi nous permettons de l'ajouter à l'ensemble des opérations. Ceci restreint les instanciations au choix d'applications linéaires inversibles, mais nous pouvons tout de même calculer les mineurs en tant que polynômes en α et α^{-1}, et il existe des instanciations MDS si et seulement si tous mes mineurs sont des polynômes non-nuls (Voir section 3.5).

3.3.4.3 Utilisation de petites puissances de α (MAX_POW).

Lorsque α est instanciée par la matrice compagnon d'un polynôme creux, α^2 a aussi un faible coût en xors, et il est donc intéressant de considérer les petites puissances de α (en particulier α^2) comme des opérations supplémentaires.

Combinée à l'extension INV, les petites puissances de α^{-1} (*e.g.* α^{-2}) sont aussi ajoutées.

3.3.4.4 Hypothèse d'opérations linéaires indépendantes (INDEP).

De façon générale, nous pouvons supposer que toutes les opérations linéaires sont indépendantes, et écrire les coefficients de la matrice comme des polynômes multivariés en α, β, γ,... Chaque opération linéaire est utilisée une seule fois, ce qui fait que les polynômes sont de degré au plus un en chaque variable. De plus, nous supposons que tous ces applications linéaires commutent afin de pouvoir tester efficacement la propriété MDS.

En pratique, nous instancions α, β, γ,... en tant que puissances d'un seul application linéaire, mais l'espace de recherche est plus petit avec cette extension qu'avec INV et MAX_POW car nous ne considérons qu'un seul application linéaire à chaque étape de l'algorithme.

Pour des raisons d'implémentation, notre code est limité à trois opérations linéaires dans ce cas (pour plus de détails, voir la section 3.3.5).

3.3.5 Choix d'implémentation

Mémoire. Dans le cadre de l'implémentation de cet algorithme, le défi principal est de gérer la mémoire. En effet, l'algorithme génère un arbre gigantesque. Chaque nœud ne consomme que très peu de mémoire (de l'ordre de 768 bits), mais le nombre de nœuds augmente exponentiellement en la taille de l'ensemble des opérations. Comme on le voit à la table 3.4, le programme requiert des centaines de gigaoctets de mémoire pour trouver des matrices 4×4, et encore plus avec certaines extensions [9].

Pour réduire la mémoire requise, nous ne stockons que le nombre minimal d'informations dans chaque nœud (son père et la dernière opération), et nous recalculons toutes les autres propriétés (*e.g.* la représentation matricielle et l'identifiant) lorsque nous en avons besoin. Ceci permet d'échanger du temps pour de la mémoire.

Test MDS. Pour tester si un nœud correspond à une matrice MDS, il nous faut construire la matrice et calculer ses mineurs. Comme mentionné auparavant, les éléments de la matrice sont des polynômes dans $\mathbb{F}_2[\alpha]$. L'implémentation vide correspond à l'identité avec des colonnes à zéro additionnelles, [10] et nous appliquons chaque opération aux colonnes de la matrice pour construire la matrice d'une implémentation donnée : la copie et le

9. Concrètement, le facteur limitant de l'algorithme est la mémoire. Avec plus de mémoire, nous aurions pu mener ces recherches plus loin.

10. Avec $k = 3$ et 4 registres, nous partons de $\left[\begin{smallmatrix} 1&0&0&0 \\ 0&1&0&0 \\ 0&0&1&0 \end{smallmatrix}\right]$

XOR correspondent aux mêmes opérations, là où l'opération linéaire α correspond à multiplier le polynôme par α.

Dans notre implémentation, nous stockons un polynôme comme un champ de bits, où chaque bit correspond à un coefficient du polynôme. En particulier, la multiplication par α correspond à un simple décalage des bits, et l'instruction de multiplication sans retenue des processeurs x86 récents (`pclmulqdq`) correspond à la multiplication de deux polynômes de degré 64 (avec une sortie de degré 128).

En pratique, nous construisons la matrice en utilisant des mots de 32 bits (*i.e.* des polynômes de degré au plus 32), et le déterminant de la matrice complète est considéré comme un polynôme de degré au plus 128 (notre code ne supporte que des matrices de taille $k \leqslant 4$). Nous calculons les mineurs avec l'expansion de Laplace, car les mineurs d'ordre petit doivent être calculés de toute manière.

Avec l'extension `INV`. Avec l'extension `INV`, il nous faut gérer des opérations α^{-1} dans le circuit. Ainsi, les coefficients de la matrice sont des polynômes en des puissances à la fois positives et négatives de α (des polynômes de Laurent). Pour notre implémentation, nous supposons que les polynômes ne contiennent que des termes entre α^{-16} et α^{15}, et nous les stockons décalés de 16 bits, *i.e.* multipliés par α^{16}. En particulier, tout le code qui teste la propriété MDS reste valide en travaillant avec ces polynômes décalés.

Avec l'extension `INDEP`. Avec l'extension `INDEP`, il nous faut gérer trois applications linéaires α, β et γ, et les coefficients de la matrice sont des polynômes multivariés dans $\mathbb{F}_2[\alpha, \beta, \gamma]$. Comme expliqué précédemment, nous supposons que chaque opération linéaire n'est utilisée qu'une fois, ce qui fait que les coefficients de la matrice sont de degré au plus un en chaque variable. Pendant le calcul des déterminants, il nous faut multiplier au plus k coefficients, donc les termes sont de degré au plus k en chaque variable. C'est pourquoi nous pouvons utiliser un encodage en tant que polynômes univariés en X, où α est encodé en X, β en X^{k+1}, γ en X^{k+1^2} et ainsi de suite. Avec cet encodage, il n'y a pas d'ambiguïté pour représenter les termes de degré au plus k en chaque variable, et la multiplication de deux polynômes univariés correspond à multiplier les polynômes multivariés correspondants.

En termes d'implémentation, ceci signifie que α correspond à un décalage de 1 bit, β de $k+1$ bits, γ de $(k+1)^2$ bits... Le reste de l'implémentation reste inchangée, et le code qui teste la propriété MDS reste valide avec cette représentation. Avec $k = 4$, la matrice contient des termes jusqu'à $\alpha\beta\gamma$, encodés en $X \cdot X^5 \cdot X^{25} = X^{31}$ et ils tiennent tous sur des mots de 32 bits.

3.4 Résultats formels

Nous avons lancé l'algorithme pour $k = 3$ et $k = 4$, en utilisant plusieurs ensembles d'extensions.

Les résultats pour $k = 3$ ont été obtenus en quelques microsecondes sur un ordinateur portable commun. Ils sont résumés à la table 3.3. La matrice la moins coûteuse, $M_{3,4}^{5,1}$, utilise 5 XORs sur $\mathbb{F}_2^k$ plus 1 opération linéaire [11]. Ce coût est notoirement inférieur au coût minimum de 6 XORs pour une implémentation naïve. À profondeur minimale (*i.e.* profondeur 2), notre meilleur résultat, $M_{3,2}^{6,3}$, prend 6 XORs sur $\mathbb{F}_2^k$ plus 3 opérations linéaires, ce qui n'est pas mieux qu'une implémentation naïve.

Pour $k = 4$ en revanche, les besoins en mémoire sont immenses : certains tests n'ont pas pu être effectués car ils demandent plus de 2.5TB de mémoire. Nous avons utilisé une machine avec 4 CPUs Intel Xeon E7-4860 v2 (48 cœurs au total) à une fréquence

11. Notez la notation $M_{3,4}^{5,1}$, qui signifie matrice 3×3 de profondeur 4, utilisant 5 XORs et 1 opération linéaire.

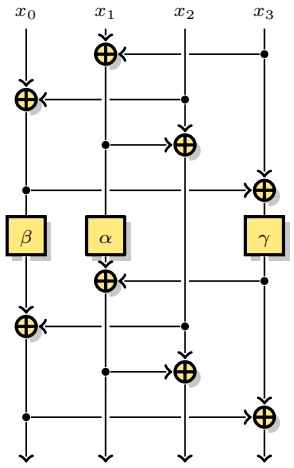


Figure 3.1 – MDS 4×4 de profondeur 5 : $M_{4,5}^{8,3}$.

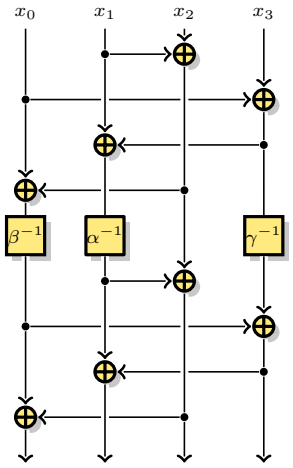


Figure 3.2 – MDS 4×4 de profondeur 5 : $M_{4,5}^{8,3^{-1}}$.

de 2.60GHz, avec un total de 2.5TB de RAM. Nous avons parallélisé le code et aucun des tests n'a pris plus de 24h en temps réel (ceux qui auraient pris plus de temps sont arrivés à court de mémoire au préalable). Nous notons que paralléliser le code est plutôt simple, puisque nous n'avons à partager que les structures qui stockent les états testés et non-testés. Les résultats les plus intéressants sont résumés à la table 3.4. La matrice la moins coûteuse, $M_{4,6}^{8,3}$, requiert 8 XORs sur $\mathbb{F}_2^k$ et 3 opérations linéaires. À profondeur 3, notre meilleur résultat, $M_{4,3}^{9,5}$, prend 9 XORs sur $\mathbb{F}_2^k$ et 5 opérations linéaires. Ces deux résultats sont significativement moins chers que le minimum de 12 XORs requis par une implémentation naïve.

Nous remarquons que nous avons en quelque sorte atteint les limites, puisque lancer l'algorithme avec $k = 4$ pour trouver des circuits de profondeur 6 à un coût moindre que la solution donnée dans la table n'a pas donné de résultats et a consommé 2.4TB de mémoire (en utilisant les extensions `RO_IN` et `INDEP`). De façon similaire, nous n'avons pas pu trouver de circuit de profondeur 3 moins coûteux que celui donné, bien que nous ayons lancé l'algorithme avec diverses extensions et limites.

Ces résultats sont des matrices formelles : l'instanciation sur $\mathbb{F}_2^4$ et $\mathbb{F}_2^8$ est discutée en section 3.5. Les figures des circuits sont données en section 3.7 (certains circuits ont été réécrits pour les rendre plus lisibles).

Implémentation de la matrice inverse. Lorsque nous implémentons l'inverse d'un chiffrement de type SPN, l'inverse de la matrice MDS est requise, et les matrices dont l'inverse peut aussi être implémentée efficacement sont les plus désirables. En particulier, un grand nombre de chiffrements à bas coût utilisent des matrices involutives, afin que la même implémentation puisse être utilisée pour le chiffrement et le déchiffrement. Notre algorithme de recherche ne nous permet pas de chercher spécifiquement des matrices involutives (ou même des matrices faciles à inverser), mais plusieurs de nos résultats ont une implémentation de l'inverse efficace.

En vérité, la plupart des matrices dans la table sont faciles à inverser car leurs registres additionnels ne servent qu'à construire des opérations du type Feistel (ce qui n'est pas le cas en général). En termes de coût d'implémentation, il est vrai que l'inverse d'une matrice a le même coût d'implémentation que la matrice directe. En termes de profondeur, en revanche, il n'y a pas de conservation entre la matrice et son inverse.

Table 3.3 – Matrices MDS 3×3 optimales (tous les résultats sont obtenus en moins d'une seconde, la mémoire est donnée en MB), où Prof. est la profondeur et Ext. est le choix d'extensions nécessaire à obtenir le résultat.

Prof.	Coût	Ext.	Mémoire	M	Fig.
4	5 XOR, 1 LIN		14	$M_{3,4}^{5,1} = \begin{bmatrix} 3 & 2 & 2 \\ 2 & 3 & 2 \\ 2 & 2 & 3 \end{bmatrix}$, $M_{3,4}^{5,1\prime} = \begin{bmatrix} 2 & 1 & 3 \\ 1 & 1 & 1 \\ 3 & 1 & 2 \end{bmatrix}$	3.4, 3.5
3	5 XOR, 2 LIN		5	$M_{3,3}^{5,2} = \begin{bmatrix} 3 & 1 & 3 \\ 1 & 1 & 2 \\ 2 & 1 & 1 \end{bmatrix}$	3.6
2	6 XOR, 3 LIN	RO_IN	4	$M_{3,2}^{6,3} = \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix}$	3.7

Pour illustrer ce phénomène, considérons l'exemple de $M_{4,5}^{8,3}$ et $M_{4,5}^{8,3^{-1}}$ représentées en figures 3.1 et 3.2 [12]. $M_{4,5}^{8,3}$ est de profondeur 5 et coûte 9 XORs et 3 opérations linéaires. Sur $\mathbb{F}_2^8$, l'instanciation décrite en section 3.5 donne que à la fois $M_{4,5}^{8,3}$ sur $\mathbb{F}_2^8$ et son inverse ont la même profondeur (en plus du même coût). Par contre, sur $\mathbb{F}_2^4$, l'instanciation de $M_{4,5}^{8,3}$ utilise A_4, A_4^{-1} et A_4^2, et donc $M_{4,5}^{8,3^{-1}}$ utilise A_4, A_4^{-1} et A_4^{-2}, données par :

$$A_4 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \qquad A_4^{-1} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \qquad A_4^2 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix} \qquad A_4^{-2} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

A_4 et A_4^{-1} ont même coût et même profondeur, mais A_4^2 ne peut s'implémenter qu'avec 2 itérations de A_4, donc une implémentation de profondeur 2, alors que A_4^{-2} possède une implémentation de profondeur 1. En résumant, sur $\mathbb{F}_2^4$, $M_{4,5}^{8,3}$ et son inverse ont le même coût, mais $M_{4,5}^{8,3}$ est de profondeur 6 alors que $M_{4,5}^{8,3^{-1}}$ est de profondeur 5.

Additionnellement, notons que certaines matrices sont presque involutives. En particulier, l'une des matrices optimales que nous avons trouvées en taille 3 est $M_{3,4}^{5,1\prime} = \left[\begin{smallmatrix} 2 & 1 & 3 \\ 1 & 1 & 1 \\ 3 & 1 & 2 \end{smallmatrix}\right]$; nous notons que son inverse est $M_{3,4}^{5,1\prime-1} = \left[\begin{smallmatrix} 3 & 1 & 2 \\ 1 & 1 & 1 \\ 2 & 1 & 3 \end{smallmatrix}\right]$, qui peut à l'évidence être calculée par le même circuit avec un simple croisement de fils supplémentaire.

Détails sur les tables. Tous les résultats donnés supposent que la profondeur de α, β et γ est 1.

Les matrices données dans ces tables sont des *exemples*. Notre intention n'était en rien d'être exhaustifs dans cette table, l'algorithme donne bien d'autres matrices formelles.

De la structure des circuits résultants. Bien que nous n'ayons trouvé que peu de structure dans les résultats, il peut être intéressant de noter que plusieurs circuits prennent la forme de réseaux de Feistel généralisés (comme définis originellement dans [Nyb96] en s'appuyant sur les travaux de Feistel, et étudiés dans de nombreux travaux depuis), à savoir les figures 3.8, 3.10, 3.13, 3.14 et 3.15.

Nous souhaitons souligner que les figures données en section 3.7 ont été intensément retouchées après la sortie de l'algorithme. Nous avons réordonné les variables d'entrée et de sortie, ainsi que certaines opérations qui commutent, afin de rendre les figures plus lisibles et de mettre en avant la structure.

12. Notez que les figures données en section 3.7 ont été réarrangées en permutant les variables d'entrée et de sortie.

Table 3.4 – Matrices MDS 4×4 optimales.

Profondeur	Coût	Extensions	Mémoire (GB)	Temps (h)	M	Fig.
6	8 XOR, 3 LIN		30.9	19.5	$M_{4,6}^{8,3} = \begin{bmatrix} 2 & 2 & 3 & 1 \\ 1 & 3 & 6 & 4 \\ 3 & 1 & 4 & 4 \\ 3 & 2 & 1 & 3 \end{bmatrix}$	3.8
5	8 XOR, 3 LIN	INDEP	24.3	2.3	$M_{4,5}^{8,3} = \begin{bmatrix} \beta & 1 & \beta+1 & 1 \\ \gamma & \alpha & \gamma & \alpha+\gamma \\ \gamma & \alpha+1 & \gamma+1 & \alpha+\gamma+1 \\ \beta+\gamma & 1 & \beta+\gamma+1 & \gamma+1 \end{bmatrix}$	3.10
5	9 XOR, 3 LIN		154.5	25.6	$M_{4,5}^{9,3} = \begin{bmatrix} 2 & 2 & 3 & 1 \\ 1 & 3 & 6 & 4 \\ 3 & 1 & 4 & 4 \\ 3 & 2 & 1 & 3 \end{bmatrix}$	3.9
4	8 XOR, 4 LIN	MAX_POW = 2	274	30.2	$M_{4,4}^{8,4} = \begin{bmatrix} 5 & 7 & 1 & 3 \\ 4 & 6 & 1 & 1 \\ 1 & 3 & 5 & 7 \\ 1 & 1 & 4 & 6 \end{bmatrix}$, $M_{4,4}^{8,4\prime} = \begin{bmatrix} 6 & 7 & 1 & 5 \\ 2 & 3 & 1 & 1 \\ 1 & 5 & 6 & 7 \\ 1 & 1 & 2 & 3 \end{bmatrix}$, $M_{4,4}^{8,4\prime\prime} = \begin{bmatrix} 3 & 2 & 1 & 3 \\ 2 & 3 & 1 & 1 \\ 4 & 3 & 6 & 4 \\ 1 & 1 & 4 & 6 \end{bmatrix}$	3.13, 3.14, 3.15
4	9 XOR, 3 LIN	INDEP	46	4.5	$M_{4,4}^{9,3} = \begin{bmatrix} \alpha+1 & \alpha & \gamma+1 & \gamma+1 \\ \beta & \beta+1 & 1 & \beta \\ 1 & 1 & \gamma & \gamma+1 \\ \alpha & \alpha+1 & \gamma+1 & \gamma \end{bmatrix}$	3.12
4	9 XOR, 4 LIN		77.7	12.8	$M_{4,4}^{9,4} = \begin{bmatrix} 1 & 2 & 4 & 3 \\ 2 & 3 & 2 & 3 \\ 3 & 3 & 5 & 1 \\ 3 & 1 & 1 & 3 \end{bmatrix}$	3.11
3	9 XOR, 5 LIN	INV	279.1	38.5	$M_{4,3}^{9,5} = \begin{bmatrix} \alpha+\alpha^{-1} & \alpha & 1 & 1 \\ 1 & \alpha+1 & \alpha & \alpha^{-1} \\ 1+\alpha^{-1} & 1 & 1 & 1+\alpha^{-1} \\ \alpha^{-1} & \alpha^{-1} & 1+\alpha^{-1} & 1 \end{bmatrix}$	3.16

En plus de cela, lorsque c'était possible, nous avons remplacé l'utilisation d'un registre additionnel par une opération de type Feistel pour faciliter la lecture.

Précisons à nouveau qu'il ne s'agit que d'exemples des sorties de l'algorithme.

Notons aussi que certaines matrices peuvent être construites par plusieurs circuits aux propriétés différentes : Les figures 3.8 et 3.9 sont identiques à réordonnancement près des sorties, mais offrent un compromis entre le coût d'implémentation et la profondeur.

3.5 Instanciation des résultats formels

3.5.1 Caractérisation des instanciations MDS

Une fois que nous avons une matrice formelle M en α avec tous les mineurs des polynômes non-nuls, nous pouvons chercher des choix concrets de α qui donnent une implémentation peu coûteuse et qui donnent une fonction linéaire de facteur de branchement maximal. Pour une matrice donnée $A \in M_n(\mathbb{F}_2)$, nous pouvons construire $M(A)$ en substituant A à α, et tester si la fonction linéaire résultante dans $M_k\big(M_n(\mathbb{F}_2)\big)$ est de facteur de branchement maximal. Comme établi en section 3.1.1.2, la fonction linéaire est de facteur de branchement maximal si et seulement si ses sous-matrices carrées en suivant les blocs de taille $n \times n$ sont non-singulières. De plus, puisque tous les blocs sont des polynômes en A, ils commutent, et nous pouvons calculer les déterminants *par blocs* [Sil00]. En effet, avec I, J des sous-ensembles des lignes et des colonnes, et $m_{I,J} = \det_{\mathbb{F}_2[\alpha]}(M_{|I,J})$ le mineur correspondant dans $\mathbb{F}_2[\alpha]$, nous avons :

$$\det_{\mathbb{F}_2}(M(A)_{|I,J}) = \det_{\mathbb{F}_2}(\det_{M_n(\mathbb{F}_2)}(M(A)_{|I,J})) = \det_{\mathbb{F}_2}(m_{I,J}(A)).$$

Ainsi, $M(A)$ est MDS si et seulement si tous les $m_{I,J}(A)$ (les mineurs formels évalués en A) sont non-singuliers.

Finalement, en posant μ_A le polynôme minimal de A (un polynôme de degré minimal tel que $\mu_A(A) = 0$), nous avons pour A la caractérisation suivante :

Proposition 3.1. *Soit $M \in M_k(\mathbb{F}_2[\alpha])$ une matrice formelle, de mineurs formels $m_{I,J}$, et $A \in M_n(\mathbb{F}_2)$ un application linéaire.*

Alors $M(A)$ est MDS si et seulement si μ_A est premier avec tous les mineurs formels $m_{I,J}$.

Démonstration. Si $\mathsf{pgcd}(\mu_A, m_{I,J}) = 1$, alors il existe deux polynômes u, v tels que $u\mu_A + vm_{I,J} = 1$ par l'identité de Bezout. En particulier

$$u(A)\mu_A(A) + v(A)m_{I,J}(A) = v(A)m_{I,J}(A) = 1,$$

donc $m_{I,J}(A)$ est non-singulière. Si cela est vrai pour tout $m_{I,J}$ alors $M(A)$ est MDS.

Réciproquement, supposons qu'il existe I, J tels que $p = \mathsf{pgcd}(\mu_A, m_{I,J})$ est non-constant. Alors $p(A)$ doit être singulière (sinon on a $\mu_A = pq$ avec $q(A) = 0$ ce qui contredit la définition du polynôme minimal μ_A). Ainsi, $m_{I,J}(A)$ est aussi singulière et $M(A)$ n'est pas MDS. □

En particulier, si tous les mineurs sont de degré strictement inférieur à n, et si π est un polynôme irréductible de degré n, alors nous pouvons utiliser la matrice compagnon de π pour A, et ceci donne une matrice MDS $M(A)$. Dans ce cas, A correspond en fait à la multiplication dans le corps fini. Plus généralement, nous pouvons utiliser cette construction même si π n'est pas irréductible. Tant que π est premier avec tous les mineurs formels $m_{I,J}$, la matrice résultante $M(A)$ sera MDS. En termes de coût d'implémentation, choisir un trinôme pour π résultera en une implémentation optimale pour l'évaluation de A : une seule portexor et des croisements de fils en matériel, ou un décalage et un xor conditionnel en logiciel.

3.5.2 Avec l'inverse

Lorsque nous utilisons aussi l'inverse de α pour construire la matrice M, les coefficients de la matrice, et les mineurs formels $m_{I,J}$, seront des polynômes de Laurent dans $\mathbb{F}_2[\alpha, \alpha^{-1}]$ plutôt que de simples polynômes. Afin d'instancier de telles matrices M, il nous faut utiliser une matrice A non-singulière, et nous avons toujours la propriété que $M(A)$ est MDS si et seulement si tous les $m_{I,J}(A)$ sont non-singuliers. De plus, nous pouvons écrire $m_{I,J} = \tilde{m}_{I,J} \times \alpha^{z_{I,J}}$ avec $\tilde{m}_{I,J}$ un polynôme ($z_{I,J}$ est choisi pour minimiser le degré de $\tilde{m}_{I,J}$), et $m_{I,J}(A)$ est non-singulier si et seulement si $\tilde{m}_{I,J}(A)$ est non-singulier, car A est nécessairement non-singulière. De ce fait, nous pouvons toujours caractériser la propriété MDS en se basant sur le polynôme minimal μ_A : $M(A)$ est MDS si et seulement si μ_A est premier avec tous les $\tilde{m}_{I,J}$.

3.5.3 Avec des multiplications indépendantes.

Lorsque nous utilisons l'extension qui donne des multiplications indépendantes, le résultat est une matrice formelle à coefficients dans $\mathbb{F}_2[\alpha, \beta, \gamma]$, dont les mineurs sont des polynômes non-nuls dans $\mathbb{F}_2[\alpha, \beta, \gamma]$. Puisque les calculs des polynômes ne font de sens que lorsque α, β et γ commutent, nous les instancierons avec des applications linéaires qui commutent. Si nous utilisons les applications A, B et C avec $AB = BA$, $AC = CA$, $BC = CB$, les polynômes évalués en A, B, C commutent, et $M(A, B, C)$ est MDS si et seulement si tous les $m_{I,J}(A, B, C)$ (les mineurs formels évalués en A, B, C) sont non-singuliers.

En particulier, si nous voulons instancier α, β et γ en tant que puissances d'un même application linéaire A, nous pouvons utiliser les résultats précédents pour caractériser les applications A qui donnent une matrice MDS à partir de leur polynôme minimal.

3.5.4 Instanciations à faible coût en xors

En pratique, nous voulons choisir A tel que $M(A)$ soit MDS, et A ait aussi un faible coût d'implémentation. En suivant les résultats de Beierle, Kranz et Leander [BKL16], nous savons que la multiplication par un élément u dans $\mathbb{F}_{2^n}$ peut être implémentée avec un seul xor binaire si et seulement si le polynôme minimal de u est un trinôme de degré n. De plus, leur preuve peut être généralisée au cas d'applications arbitraires A dans $M_n(\mathbb{F}_2)$, avec le résultat suivant : si A peut être implémenté avec un seul xor binaire, alors soit A est singulière (*i.e.* $\alpha | \mu_A$), soit $A + 1$ est singulière (*i.e.* $(\alpha + 1) | \mu_A$), soit μ_A est un trinôme de degré n.

Comme toutes les matrices que nous listons dans les tables 3.3 et 3.4 ont α et $\alpha + 1$ pour mineurs, les seuls candidats intéressants avec un décompte de xors égal à 1 sont les matrices dont le polynôme minimal est un trinôme de degré n. C'est pourquoi nous concentrons nos recherches sur les matrices compagnons de trinômes. (Pour un trinôme t donné, il y a beaucoup de matrices différentes avec un décompte de xors égal à 1 et t comme polynôme minimal, mais elles sont soit toutes MDS, soit toutes non-MDS, du fait de la proposition 3.1.)

3.5.5 Choix concrets d'instanciations

Nous instancions maintenant les matrices de la table 3.1. Nous définissons A_8 la matrice compagnon de $X^8 + X^2 + 1$ sur $\mathbb{F}_2$; A_8^{-1} a pour polynôme minimal $X^8 + X^6 + 1$:

$$A_8 = \begin{bmatrix} 0&0&0&0&0&0&0&1 \\ 1&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&1 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&1&0 \end{bmatrix} \qquad A_8^{-1} = \begin{bmatrix} 0&1&0&0&0&0&0&0 \\ 1&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&1 \\ 1&0&0&0&0&0&0&0 \end{bmatrix}$$

Similairement, nous définissons A_4 la matrice compagnon de $X^4 + X + 1$ sur $\mathbb{F}_2$; A_4^{-1} a pour polynôme minimal $X^4 + X^3 + 1$:

$$A_4 = \begin{bmatrix} 0&0&0&1 \\ 1&0&0&1 \\ 0&1&0&0 \\ 0&0&1&0 \end{bmatrix} \qquad A_4^{-1} = \begin{bmatrix} 1&1&0&0 \\ 0&0&1&0 \\ 0&0&0&1 \\ 1&0&0&0 \end{bmatrix}$$

Bien que ce ne soit pas le cas de manière générale, pour les matrices de la table 3.1, A_8, A_4, A_8^{-1} et A_4^{-1} suffisent à instancier les résultats de l'algorithme sur $\mathbb{F}_2^8$. Par exemple, sur $\mathbb{F}_2[X]$:

Les trinômes et leurs factorisations sont

$$\begin{aligned}
X^8 + X \ \ + 1 &= (X^2 + X + 1)(X^6 + X^5 + X^3 + X^2 + 1), \\
X^8 + X^2 + 1 &= (X^4 + X + 1)^2, \\
X^8 + X^3 + 1 &= (X^3 + X + 1)(X^5 + X^3 + X^2 + X + 1), \\
X^8 + X^4 + 1 &= (X^2 + X + 1)^4, \\
X^8 + X^5 + 1 &= (X^3 + X^2 + 1)(X^5 + X^4 + X^3 + X^2 + 1), \\
X^8 + X^6 + 1 &= (X^4 + X^3 + 1)^2, \\
X^8 + X^7 + 1 &= (X^2 + X + 1)(X^6 + X^4 + X^3 + X + 1).
\end{aligned}$$

En particulier, il y a seulement 2 trinômes qui se factorisent en polynômes de degré 4 : $X^8 + X^2 + 1 = (X^4 + X + 1)^2$ et $X^8 + X^6 + 1 = (X^4 + X^3 + 1)^2$.

Les mineurs de $\boldsymbol{M_{4,6}^{8,3}} = \begin{bmatrix} 2&2&3&1 \\ 1&3&6&4 \\ 3&1&4&4 \\ 3&2&1&3 \end{bmatrix}$ **sont**

$\{1, X, X+1, X^2, X^2+1, X^2+X, X^2+X+1, X^3, X^3+1, X^3+X, X^3+X+1, X^3+X^2+1, X^3+X^2+X, X^3+X^2+X+1\}$
dont les facteurs sont

$$\{X, X+1, X^3+X+1, X^2+X+1, X^3+X^2+1\}.$$

Aucun n'est de degré plus grand que 3, aussi ils sont tous premiers à la fois avec $X^8 + X^2 + 1$ et $X^8 + X^6 + 1$. Sélectionner soit $\alpha = A_8$, soit $\alpha = A_8^{-1}$ donne donc une matrice MDS sur $\mathbb{F}_2^8$. De plus, choisir A_8^{-1} permet de réduire la profondeur à 5, car A^{-2} peut être implémentée à profondeur 1. Une implémentation complète est donnée en section 3.8.2.

Les facteurs des mineurs de $\boldsymbol{M_{4,4}^{8,4}} = \begin{bmatrix} 5&7&1&3 \\ 4&6&1&1 \\ 1&3&5&7 \\ 1&1&4&6 \end{bmatrix}$ **sont**

$$\{X, X+1, X^3+X+1, X^2+X+1, X^3+X^2+1, X^4+X^3+1\}.$$

L'unique facteur de degré 4 est $X^4 + X^3 + 1$, donc il y a au moins un mineur qui n'est pas premier avec $X^8 + X^6 + 1$, mais ils sont tous premiers avec $X^8 + X^2 + 1$. Sélectionner $\alpha = A_8$ donne ainsi une matrice MDS sur $\mathbb{F}_2^8$.

Les autres résultats sont obtenus de façon similaire.

3.5.5.1 Instanciation de $\boldsymbol{M_{4,5}^{8,3}} = \begin{bmatrix} \beta & 1 & \beta+1 & 1 \\ \gamma & \alpha & \gamma & \alpha+\gamma \\ \gamma & \alpha+1 & \gamma+1 & \alpha+\gamma+1 \\ \beta+\gamma & 1 & \beta+\gamma+1 & \gamma+1 \end{bmatrix}$

Suivant la section 3.5.3, nous instancions tous les applications linéaires en tant que puissances d'un unique α. En utilisant le code `sage` donné en section 3.8.1, nous trouvons que fixer $\beta = \alpha^{-1}$ et $\gamma = \alpha^2$ donne toujours une matrice MDS. Les facteurs des mineurs des matrices résultantes sont :

$$X, X+1, X^2+X+1, X^3+X+1, X^3+X^2+1, X^4+X+1$$

Le seul facteur de degré 4 est X^4+X^3+1, donc $\alpha = A^{-1}$ donne une matrice MDS sur $\mathbb{F}_2^8$.

3.6 Conclusion sur les constructions de matrices de diffusion à bas coût

Tout comme les travaux de [Kra+17], nos résultats montrent que l'optimisation globale d'une matrice MDS est bien plus puissante que l'optimisation locale des coefficients. De plus, notre approche permet de trouver de nouvelles matrices MDS optimisées pour une implémentation globale à bas coût, alors que les straight line programs de [Kra+17] ne peuvent que trouver une bonne implémentation d'une matrice donnée. Comme le montre la table 3.1, notre approche amène des résultats meilleurs encore. En particulier, nous donnons une matrice 4×4 MDS avec des mots de taille 8 bit, de décompte de xors 67, alors que le meilleur résultat précédent était 72.

Qui plus est, notre approche peut prendre en compte la profondeur des circuits. Lorsque nous considérons des résultats à profondeur 3 (la profondeur minimale atteignable), nous avons encore des matrices MDS avec un décompte de xors de seulement 77, ce qui est compétitif par rapport aux résultats des optimisations par straight line programs.

Enfin, nous avons essayé d'optimiser nos résultats en y appliquant des outils de synthèse de circuit, et de straight line programs sur les matrices binaires correspondantes, mais ces approches n'ont pas donné de meilleurs circuits (cf. table 3.5).

Recherche de matrices presque-MDS. Les travaux présentés dans ce chapitre se concentrent sur la recherche de matrices MDS, or il peut être plus adapté de choisir des matrices presque-MDS (de facteur de branchement différentiel et linéaire k pour une matrice $k \times k$) suivant le cas d'usage (cf. [Kho+14]). À la suite d'une suggestion de Gregor Leander, nous avons adapté notre algorithme pour chercher des matrices presque-MDS.

Deux modifications s'imposent : changer le test pour arrêter l'algorithme lorsqu'on détecte une matrice presque-MDS (et non plus MDS) et adapter l'heuristique A^*.

Par contrainte de temps, nous d'avons pas adapté A^*, et l'avons simplement désactivé. Tester si une matrice est presque-MDS rapidement demande une caractérisation des matrices presque-MDS, nous utilisons la suivante : une matrice $k \times k$ est presque-MDS si pour tout b, $1 \leqslant b \leqslant k-1$, toute sous-matrice rectangulaire $b \times (b+1)$ ou $(b+1) \times b$ est de rang b. Ceci se calcule directement à partir des mineurs de taille $b \times b$ pour tout b, $1 \leqslant b \leqslant k-1$, tout comme le test MDS.

Les résultats que nous avons obtenus ne sont malheureusement pas surprenants. En taille 3×3, la meilleure matrice presque-MDS (dans notre espace de recherche) est

$$\begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix},$$

qui s'implémente sans difficulté à profondeur 1 avec 3 XORs sur les mots [13]. En taille 4×4, le meilleur résultat est une matrice bien connue,

$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix},$$

13. Chaque mot de sortie est le XOR de 2 mots d'entrée.

qui peut s'implémenter à profondeur 3 avec 6 XORs tel que le montre la figure 3.3a. Une implémentation à profondeur 2 existe (et nous a été mentionnée par Eli Biham), qui utilise 2 registres supplémentaires comme le montre la figure 3.3b.

Ceci permet de donner l'écart entre la meilleure matrice MDS connue et la meilleure matrice presque-MDS connue.

Notons que la recherche est significativement plus rapide et moins coûteuse en mémoire que celle des matrices MDS, aussi il paraît envisageable de trouver les meilleures matrices presque-MDS de taille 5×5. Nous avons tenté cette recherche, mais l'algorithme ne trouve pas de telle matrice tel quel[14]. En revanche, en adaptant l'heuristique A^* au cas presque-MDS plutôt que de la désactiver, il paraît envisageable de pousser plus loin ces recherches.

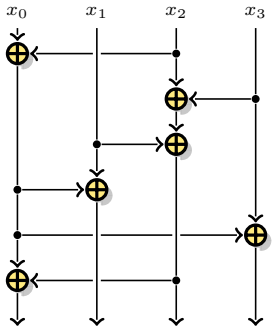


Figure 3.3(a): Matrice 4×4 presque-MDS à profondeur 3.

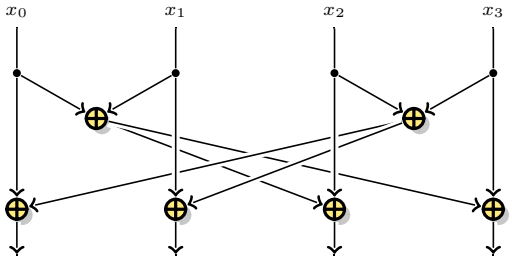


Figure 3.3(b): Matrice 4×4 presque-MDS à profondeur 2.

3.7 Résultats en figures

3.7.1 Matrices formelles de taille 3×3

3.7.2 Matrices formelles de taille 4×4

14. Nous avons pu tester pour des circuits jusqu'à 12 XORs.

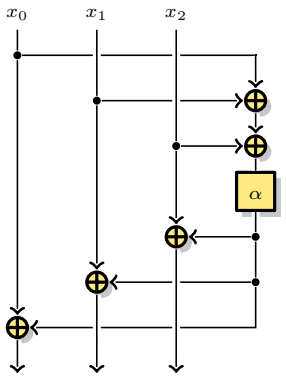


Figure 3.4 – Matrice MDS 3×3 de profondeur 4 : $M_{3,4}^{5,1} = \begin{bmatrix} 3 & 2 & 2 \\ 2 & 3 & 2 \\ 2 & 2 & 3 \end{bmatrix}$

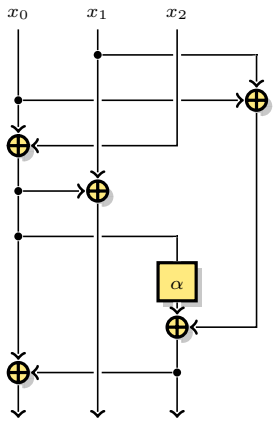


Figure 3.5 – Matrice MDS 3×3 de profondeur 4 : $M_{3,4}^{5,1'} = \begin{bmatrix} 2 & 1 & 3 \\ 1 & 1 & 1 \\ 3 & 1 & 2 \end{bmatrix}$

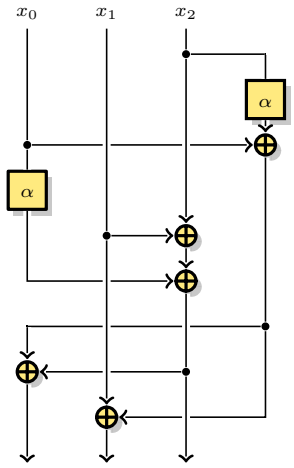


Figure 3.6 – Matrice MDS 3×3 de profondeur 3 : $M_{3,3}^{5,2} = \begin{bmatrix} 3 & 1 & 3 \\ 1 & 1 & 2 \\ 2 & 1 & 1 \end{bmatrix}$

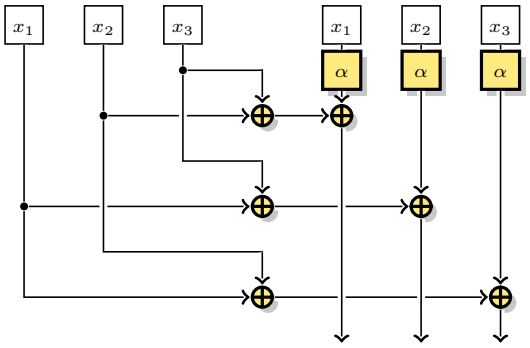


Figure 3.7 – Matrice MDS 3×3 de profondeur 2 : $M_{3,2}^{6,3} = \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix}$

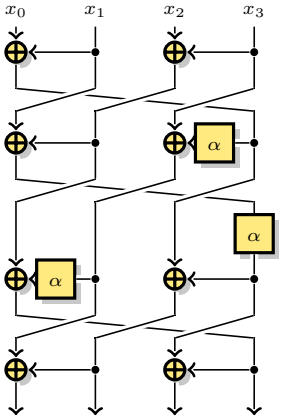


Figure 3.8 – Matrice MDS 4×4 de profondeur 6 : $M_{4,6}^{8,3} = \begin{bmatrix} 3 & 1 & 4 & 4 \\ 1 & 3 & 6 & 4 \\ 2 & 2 & 3 & 1 \\ 3 & 2 & 1 & 3 \end{bmatrix}$

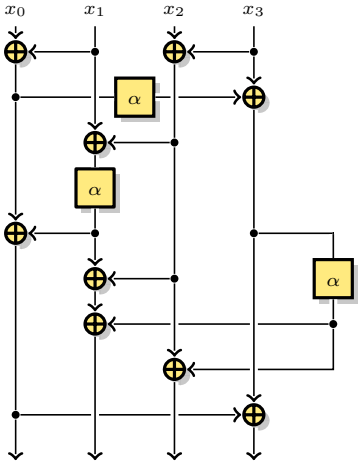


Figure 3.9 – Matrice MDS 4×4 de profondeur 5 : $M_{4,5}^{9,3} = \begin{bmatrix} 2 & 2 & 3 & 1 \\ 1 & 3 & 6 & 4 \\ 3 & 1 & 4 & 4 \\ 3 & 2 & 1 & 3 \end{bmatrix}$

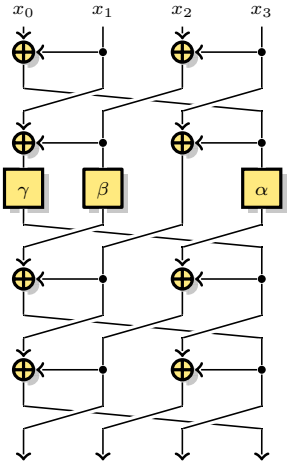


Figure 3.10 – Matrice MDS 4×4 de profondeur 5 : $M_{4,5}^{8,3} = \begin{bmatrix} \alpha & \alpha+\gamma & \gamma & \gamma \\ 1 & \gamma+1 & \gamma+\beta & \gamma+\beta+1 \\ 1 & 1 & \beta & \beta+1 \\ \alpha+1 & \alpha+\gamma+1 & \gamma & \gamma+1 \end{bmatrix}$

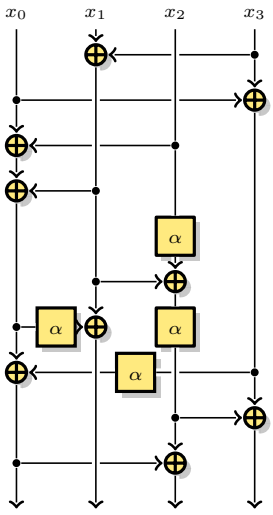


Figure 3.11 – Matrice MDS 4×4 de profondeur 4 : $M_{4,4}^{9,4} = \begin{bmatrix} 1 & 2 & 4 & 3 \\ 2 & 3 & 2 & 3 \\ 3 & 3 & 5 & 1 \\ 3 & 1 & 1 & 3 \end{bmatrix}$

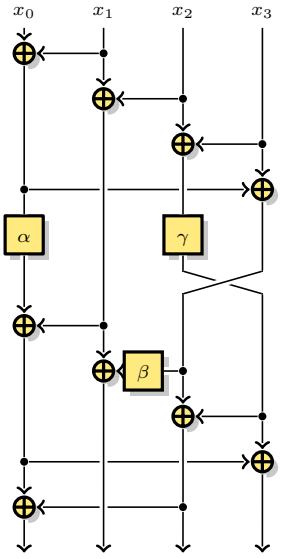


Figure 3.12 – Matrice MDS 4×4 de profondeur 4 : $M_{4,4}^{9,3} = \begin{bmatrix} \alpha+1 & \alpha & \gamma+1 & \gamma+1 \\ \beta & \beta+1 & 1 & \beta \\ 1 & 1 & \gamma & \gamma+1 \\ \alpha & \alpha+1 & \gamma+1 & \gamma \end{bmatrix}$

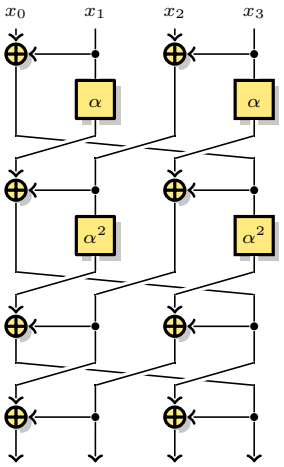


Figure 3.13 – Matrice MDS 4×4 de profondeur 4 : $M_{4,4}^{8,4} = \begin{bmatrix} 5 & 7 & 1 & 3 \\ 4 & 6 & 1 & 1 \\ 1 & 3 & 5 & 7 \\ 1 & 1 & 4 & 6 \end{bmatrix}$ avec $\alpha = 2$.

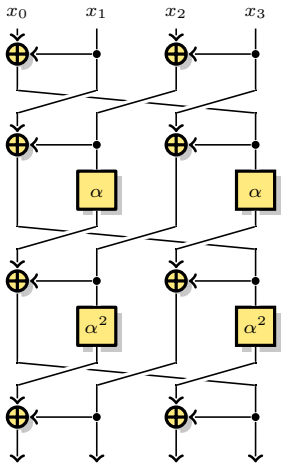


Figure 3.14 – Matrice MDS 4×4 de profondeur 4 : $M_{4,4}^{8,4'} = \begin{bmatrix} 6 & 7 & 1 & 5 \\ 2 & 3 & 1 & 1 \\ 1 & 5 & 6 & 7 \\ 1 & 1 & 2 & 3 \end{bmatrix}$ avec $\alpha = 2$ ($\alpha \leftrightarrow \alpha^2$ is also MDS).

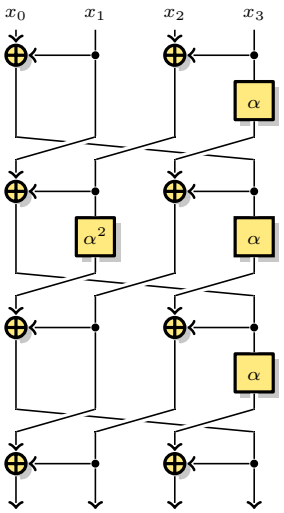


Figure 3.15 – Matrice MDS 4×4 de profondeur 4 $M_{4,4}^{8,4''} = \begin{bmatrix} 3 & 2 & 1 & 3 \\ 2 & 3 & 1 & 1 \\ 1 & 3 & 6 & 4 \\ 1 & 1 & 4 & 6 \end{bmatrix}$ avec $\alpha = 2$ ($\alpha \leftrightarrow \alpha^2$ is also MDS).

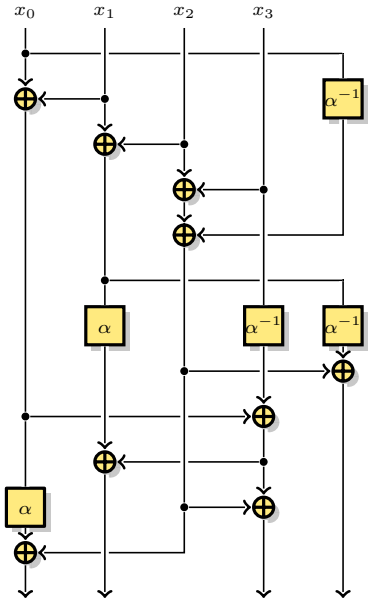


Figure 3.16 – Matrice MDS 4×4 de profondeur 3 : $M_{4,3}^{9,5} = \begin{bmatrix} \alpha+\alpha^{-1} & \alpha & 1 & 1 \\ 1 & \alpha+1 & \alpha & \alpha^{-1} \\ 1+\alpha^{-1} & 1 & 1 & 1+\alpha^{-1} \\ \alpha^{-1} & \alpha^{-1} & 1+\alpha^{-1} & 1 \end{bmatrix}$

3.8 Code utile

3.8.1 Code sage

Le programme sage suivant permet d'instancier les constructions de la section 3.5.

```
R.<a,b,c> = PolynomialRing(GF(2))
M = Matrix([[b,1,b+1,1], \
            [c,a,c,a+c], \
            [c,a+1,c+1,a+c+1], \
            [b+c,1,b+c+1,c+1]])
#M = Matrix([[a,a,a+1,1], \
#            [1,a+1,a*(a+1),a^2], \
#            [a+1,1,a^2,a^2], \
#            [a+1,a,1,a+1]])
#M = Matrix([[a^2+1,a^2+a+1,1,a+1], \
#            [a^2,a^2+a,1,1], \
#            [1,a+1,a^2+1,a^2+a+1], \
#            [1,1,a^2,a^2+a]])
#M = Matrix([[a+1,a,1,a+1], \
#            [a,a+1,1,1], \
#            [1,a+1,a^2+a,a^4], \
#            [1,1,a^2,a^2+a]])
can_invert = lambda m: m.is_invertible() \
             if hasattr(m,"is_invertible") \
                else not m.is_zero()
all_minors = lambda M : [ m for k in range (M.nrows()) \
                            for m in M.minors(k+1) ]
minors_factor = lambda M : { p for m in all_minors(M) \
                               for p,_ in factor(m)}
is_MDS = lambda M : all(can_invert(m) for m in all_minors(M))
print is_MDS(M)

MS = MatrixSpace(GF(2),8,8)
MS.is_field = lambda proof=True: False
A_8 = MS([[0,1,0,0,0,0,0,0],[0,0,1,0,0,0,0,0], \
            [0,0,0,1,0,0,0,0],[0,0,0,0,1,0,0,0], \
            [0,0,0,0,0,1,0,0],[0,0,0,0,0,0,1,0], \
            [0,0,0,0,0,0,0,1],[1,0,1,0,0,0,0,0]])
A_8 = A_8^-1
print is_MDS(M.substitute({a:A_8,b:A_8^-1,c:A_8^-2}))
```

3.8.2 Code C

La meilleure matrice MDS 4×4 obtenue sur des mots de 8 bits peut être implémentée en 67 xors binaires. Nous l'obtenons à partir de $M_{4,6}^{8,3}$ avec $\alpha = A_8$. Ceci correspond à la

matrice binaire suivante :

$$M_{4,6}^{8,3}(A_8) = \begin{bmatrix} A_8 & A_8 & A_8 \oplus I & I \\ I & A_8 \oplus I & A_8^2 \oplus A_8 & A_8^2 \\ A_8 \oplus I & I & A_8^2 & A_8^2 \\ A_8 \oplus I & A_8 & I & A_8 \oplus I \end{bmatrix}$$

$$= \left[\begin{array}{cccccccc|cccccccc|cccccccc|cccccccc}
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
0&1&0&0&0&0&0&1&0&1&0&0&0&0&0&1&0&1&1&0&0&0&0&1&0&0&1&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&1\\
\hline
1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&1&1&0&0&0&0&0&0&1&0\\
0&1&0&0&0&0&0&0&1&1&0&0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1\\
0&0&1&0&0&0&0&0&0&1&1&0&0&0&0&1&1&1&0&0&0&0&1&1&1&0&0&0&0&0&1&0\\
0&0&0&1&0&0&0&0&0&0&1&1&0&0&0&0&0&1&1&0&0&0&0&1&0&1&0&0&0&0&0&1\\
0&0&0&0&1&0&0&0&0&0&0&1&1&0&0&0&0&0&1&1&0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&1&1&0&0&0&0&0&1&1&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&1&1&0&0&0&0&0&1&1&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&1&0&0&0&0&0&1&1&0&0&0&0&0&0&1&0&0\\
\hline
1&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&0&0\\
1&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1\\
0&1&1&0&0&0&0&1&0&0&1&0&0&0&0&0&1&0&0&0&0&0&1&0&1&0&0&0&0&1&0&0\\
0&0&1&1&0&0&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&1&0&1&0&0&0&0&1&0\\
0&0&0&1&1&0&0&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&1&1&0&0&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&1&1&0&0&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0\\
\hline
1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1\\
1&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&1&0&0&0&0&0&0\\
0&1&1&0&0&0&0&1&0&1&0&0&0&0&0&1&0&0&1&0&0&0&0&0&0&1&1&0&0&0&0&1\\
0&0&1&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&1&0&0&0&0\\
0&0&0&1&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&1&0&0&0\\
0&0&0&0&1&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&1&0&0\\
0&0&0&0&0&1&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&1&0\\
0&0&0&0&0&0&1&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&1
\end{array}\right]$$

Le code `C` suivant permet d'implémenter cette matrice. Les décalages dans la figure sont implicites dans ce code, et la fonction linéaire `LIN` correspond à A_8. Les autres matrices peuvent être implémentées de façon similaire.

```
#define ROT(x) (((x)>>7) | ((x)<<1))
#define LIN(x) (ROT(x) ^ (((x&0x80)>>5)))

uint32_t MDS(uint32_t x) {
  uint8_t a = x, b = x>>8, c = x>>16, d = x>>24;
  a ^= b;
  c ^= d;
  d ^= LIN(a);
  b ^= c;
  b  = LIN(b);
  a ^= b;
  c ^= LIN(d);
  d ^= a;
  b ^= c;
  return ((((((uint32_t)c<<8) | b)<<8) | a)<<8) | d;
}
```

3.9 Application des outils d'optimisation sur nos meilleurs résultats

Nous avons tenté d'optimiser nos meilleurs résultats en utilisant les outils d'optimisation de matrices de [BP10] et de [Paa97] en adaptant le code de [Kra+17], et d'optimisation de circuits Yosys. Les résultats sont donnés en table 3.5.

Notons que Yosys part d'un circuit, et tente donc d'optimiser nos circuits, alors que les autres outils partent d'une matrice (et ne peuvent donc pas réutiliser nos circuits).

Dans tous les cas, ces outils d'optimisation cherchent des circuits qui ne font pas partie de notre espace de recherche, à savoir des circuits utilisant des opérations binaires plutôt que des opérations sur des mots. Surprenamment, même en autorisant ces opérations binaires, aucun de ces outils n'est capable d'améliorer nos résultats d'implémentation.

Ici, nous avons testé toutes les matrices que nous donne l'algorithme lorsque plusieurs matrices optimales étaient trouvées. Nous ne donnons dans la table que les résultats pour celles citées dans la table 3.4, mais les résultats sont similaires pour les autres matrices trouvées par l'algorithme.

Table 3.5 – Instanciation de certains de nos résultats et comparaison de nos circuits avec plusieurs outils d'optimisation : Yosys et les heuristiques de [BP10] et [Paa97] (avec une limite de 3 heures). Nous donnons le nombre de xors et la profondeur du circuit.

Matrix	Inst.	Nôtre	Yosys	BP	Paar1	Paar2	Naïve
$M_{4,5}^{9,3}$	$\alpha = A_4$	39/5	35/5	38/7	46/4	45/5	87/4
$M_{4,5}^{9,3}$	$\alpha = A_4^{-1}$	39/5	36/5	40/4	46/4	46/4	77/3
$M_{4,6}^{8,3}$	$\alpha = A_4$	35/5	35/5	38/7	46/4	45/5	87/4
$M_{4,6}^{8,3}$	$\alpha = A_4^{-1}$	35/6	35/5	40/4	46/4	46/4	77/3
$M_{4,5}^{8,3}$	$\alpha = A_4^{-1}, \beta = A_4, \gamma = A_4^{-2}$	36/6	36/6	40/6	48/4	47/4	99/4
$M_{4,4}^{9,4}$	$\alpha = A_4$	40/4	39/4	41/9	49/5	47/5	94/4
$M_{4,4}^{9,3}$	$\alpha = A_4, \beta = A_4^{-1}, \gamma = A_4^2$	40/4	40/4	40/7	45/4	43/4	93/4
$M_{4,4}^{8,4}$	$\alpha = A_4$	38/4	38/4	40/7	43/5	43/5	102/4
$M_{4,4}^{8,4'}$	$\alpha = A_4$	38/4	38/4	43/6	41/4	41/4	100/4
$M_{4,4}^{8,4''}$	$\alpha = A_4$	37/4	37/4	40/5	44/4	43/5	84/4
$M_{4,3}^{9,5}$	$\alpha = A_4$	41/3	41/3	40/4	44/4	43/4	73/3
$M_{4,3}^{9,5}$	$\alpha = A_4^{-1}$	41/3	41/3	43/5	44/3	44/3	77/3
$M_{4,5}^{9,3}$	$\alpha = A_8$	75/5	67/5	74/5	88/4	88/4	161/4
$M_{4,5}^{9,3}$	$\alpha = A_8^{-1}$	75/5	67/5	71/6	89/5	89/5	161/4
$M_{4,6}^{8,3}$	$\alpha = A_8$	67/5	67/5	74/5	88/4	88/4	161/4
$M_{4,6}^{8,3}$	$\alpha = A_8^{-1}$	67/5	67/5	71/6	89/5	89/5	161/4
$M_{4,5}^{8,3}$	$\alpha = A_8^{-1}, \beta = A_8, \gamma = A_8^{-2}$	68/5	68/5	75/6	81/4	77/4	186/4
$M_{4,4}^{9,4}$	$\alpha = A_8$	76/4	76/4	77/6	92/4	92/4	174/4
$M_{4,4}^{9,3}$	$\alpha = A_8, \beta = A_8^{-1}, \gamma = A_8^2$	76/4	76/4	76/6	83/6	83/6	171/4
$M_{4,4}^{8,4}$	$\alpha = A_8$	70/4	70/4	72/5	76/4	74/4	198/4
$M_{4,4}^{8,4'}$	$\alpha = A_8$	70/4	70/4	81/7	79/5	79/5	196/4
$M_{4,4}^{8,4''}$	$\alpha = A_8$	69/4	69/4	72/6	85/5	85/5	160/4
$M_{4,3}^{9,5}$	$\alpha = A_8$	77/3	77/3	76/7	86/4	86/4	145/3
$M_{4,3}^{9,5}$	$\alpha = A_8^{-1}$	77/3	77/3	79/5	86/4	86/4	145/3

Chapitre 4

Cryptanalyse

Il faut sauver FLIP

Ce chapitre est dédié à un aspect très différent de la cryptographie symétrique : non seulement il ne s'agit pas de construction, mais de cryptanalyse (qui a engendré de nouvelles perspectives de construction), mais il ne s'agit pas non plus de chiffrements par blocs.

Nous nous intéresserons ici à FLIP [Méa+16], une famille de chiffrements à flot à bas coût (pour une notion de bas coût très spécifique), conçue spécialement pour répondre au besoin du chiffrement complètement homomorphe (Fully Homomorphic Encryption, abrégé FHE).

Afin de réduire le nombre de multiplications effectuées pour chiffrer un message, les auteurs de FLIP ont drastiquement diminué le nombre de termes de haut degré algébrique dans l'expression de leur chiffrement. Dans notre article de Crypto 2016 [DLR16], Virginie Lallemand, Yann Rotella et moi-même avons identifié ce choix comme une faiblesse et l'avons exploitée pour obtenir une attaque efficace sur les chiffrements de la famille FLIP (tant en théorie qu'en pratique, puisque nous avons implémenté l'attaque et avons pu vérifier notre modèle théorique).

Les conséquences de notre attaque. Précisons tout de même que, par FLIP, nous entendrons ici la version originale de FLIP (présentée en séminaire national aux Journées Codes et Cryptographie de 2015). En effet, la version officiellement publiée de FLIP (à Eurocrypt 2016 [Méa+16]) est une version mise à jour qui prend en compte notre cryptanalyse et la rend impraticable (voir section 4.7).

On notera que, à la suite de cette cryptanalyse, Claude Carlet, Pierrick Méaux et Yann Rotella ont développé dans [CMR17] une analyse plus poussée des chiffrements utilisant des choix de construction spécifiques tels que FLIP. Ils ont déterminé des critères de construction permettant de justifier correctement la sécurité de ce chiffrement.

Organisation. Cette section est divisée en deux grandes parties : dans un premier temps, nous donnons une description des besoins exprimés par les spécialistes du FHE et de FLIP, le chiffrement symétrique adapté au FHE conçu par Méaux, Journault, Standaert et Carlet.

Dans un second temps, nous identifions la faiblesse de FLIP et l'exploitons pour monter une attaque algébrique via une analyse probabiliste de type guess-and-determine (supposer et déterminer). Nous montrons la complexité théorique de l'attaque et, comme cette attaque est faisable en temps pratique, nous l'implémentons et pouvons vérifier que la pratique colle parfaitement à notre analyse théorique.

4.1 Le pari fou du FHE

4.1.1 Le problème de départ

Le FHE (Chiffrement Complètement Homomorphe) est une solution récente à une problématique importante de la cryptographie : déléguer des calculs coûteux à un agent tiers, sans que ledit agent ne récupère d'information sur les données qu'il manipule.

Typiquement, on considère qu'un usager confie ses données médicales au cloud et demande au cloud de calculer une certaine fonction de ses données, comme par exemple les risques que l'usager ait un cancer.

À l'évidence, dans le respect de la vie privée, il est essentiel qu'une telle délégation des calculs ne révèle pas les données médicales de l'usager au cloud et que le cloud ne sache pas non plus quel est le risque de cancer de l'usager. La solution serait donc que toutes les données manipulées, ainsi que le résultat des calculs, soient chiffrés à chaque instant.

On remarquera que, l'agent tiers (le cloud) n'étant pas de confiance, on ne lui confiera pas de clef secrète, ce qui implique qu'on utilisera du chiffrement asymétrique : l'usager possède sa clef secrète et est le seul à savoir déchiffrer.

4.1.2 La solution du FHE

4.1.2.1 La solution initiale

Le problème est bien sûr que le cloud devrait alors effectuer des calculs sur des données chiffrées afin d'obtenir un résultat correct sur les données en clair. Ceci semble insensé, et pourtant en 2009, Craig Gentry a démontré qu'un schéma de chiffrement permettant de calculer à partir des données chiffrées n'importe quelle fonction des données en clair est réalisable (bien que sa solution, trop coûteuse, tienne plutôt de la preuve de concept).

L'aspect essentiel sur lequel se concentre Gentry est l'homomorphisme : pour qu'un chiffrement permette ces bonnes propriétés, il suffit qu'il s'agisse d'un *chiffrement complètement homomorphe*, c'est-à-dire qui conserve la structure des données.

Le chiffrement homomorphe n'étant pas l'objet de ce travail, nous ne décrirons pas plus loin dans les détails le chiffrement homomorphe. Nous nous contenterons de dire que depuis, plusieurs générations de chiffrement complètement homomorphe ont émergé (les différences entre génération se concentrant sur la métrique de coût) et que des propositions de chiffrement de plus en plus proches d'être utilisables en pratique ont été développées. Un aspect récurrent dans tous ces chiffrements est le concept de *bruit*.

4.1.2.2 FHE à coût raisonnable : le chiffrement homomorphe hybride

Il est supposé en chiffrement complètement homomorphe que Alice envoie au cloud Claude ses données chiffrées à l'aide d'un chiffrement homomorphe. Or, en utilisant exclusivement des chiffrements asymétriques, toutes les solutions proposées jusqu'ici entraînent une large expansion du chiffré, c'est-à-dire un message chiffré sur beaucoup plus de bits que le message clair, ce qui fait que cette transmission de données initiale est très coûteuse.

Réduire la taille du chiffré est un problème important qui a été considéré pour la première fois dans [NLV11]. Une solution proposée est d'utiliser un chiffrement symétrique : les données sont transmises *chiffrées par un chiffrement symétrique* (sans expansion du chiffré), puis déchiffrées homomorphiquement par le cloud, qui est capable de retrouver un chiffré des données *par un chiffrement homomorphe*[1]. On appelle cette technique le *chiffrement homomorphe hybride*.

1. Le chiffrement homomorphe étant asymétrique, Claude peut chiffrer homomorphiquement le chiffré symétrique reçu d'Alice, puis, en utilisant un chiffré homomorphe de la clef de déchiffrement symétrique, il peut appliquer la fonction de déchiffrement symétrique homomorphiquement pour récupérer un chiffré des données par un chiffrement asymétrique.

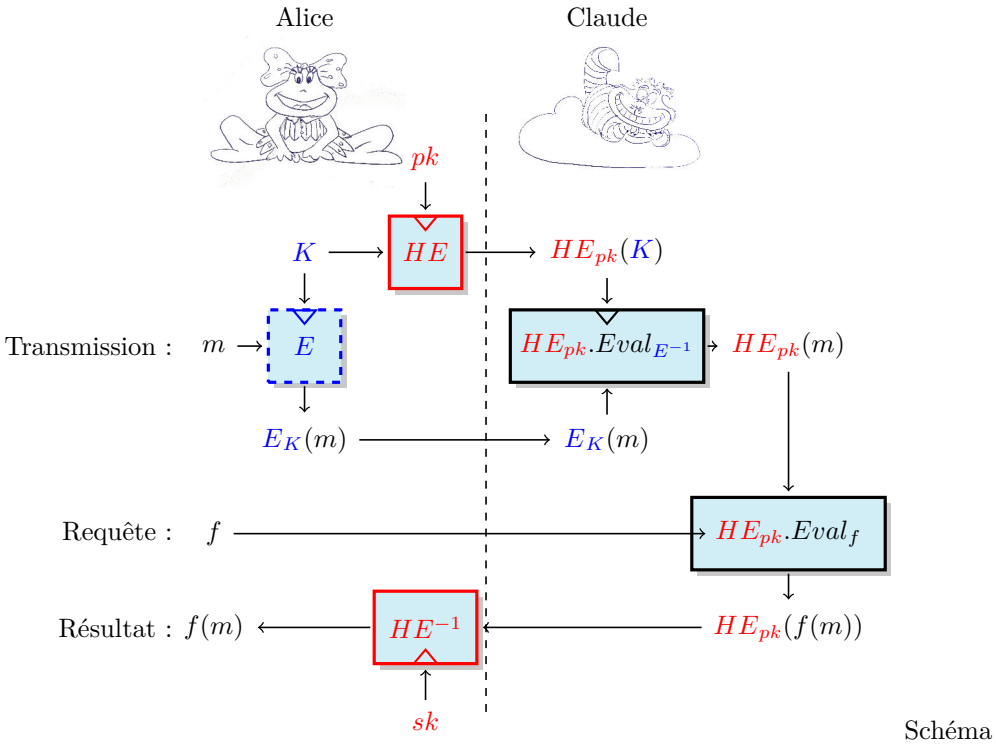


Schéma général d'un protocole de délégation de calcul sans révéler d'information grâce au chiffrement homomorphe.

Les informations secrètes sont le message m, la clef de [dé]chiffrement symétrique K et la clef de déchiffrement asymétrique sk. $HE_{pk}.Eval_f$ représente l'évaluation homomorphe d'une fonction f.
Les fonctions et clefs symétriques sont en bleu, les asymétriques en rouge.
Ce protocole est séparé en 2 phases : la phase de transmission du message m, puis une ou plusieurs phases de requêtes où Alice demande à Claude de calculer des opérations sur ses données.
On remarque que :

- les informations secrètes n'apparaissent jamais en clair durant les échanges, ni chez le cloud Claude,
- le seul surcoût pour Claude en plus de l'évaluation de la requête f est l'évaluation homomorphe du déchiffrement symétrique E^{-1}. Minimiser le bruit apporté par l'évaluation homomorphe de E^{-1} garantit donc que le l'évaluation de f commencera avec une entrée peu bruitée,
- on ne chiffre jamais le message avec un chiffrement asymétrique (ce qui évite de manipuler de grands chiffrés puisque pour un chiffrement asymétrique les chiffrés sont significativement plus grands que les clairs),
- On n'a besoin d'effectuer la transmission qu'une seule fois, même si l'on fait plusieurs requêtes,
- Le chiffrement symétrique n'intervient que dans la phase de transmission.

L'intérêt est que les données à transmettre sont alors : les données chiffrées par un chiffrement symétrique (sans expansion du chiffré) et la clef de chiffrement symétrique (petite et de taille fixe), chiffrée par un chiffrement homomorphe (avec expansion du chiffré, mais une expansion contrôlée et raisonnable du fait de la taille de clef fixée et petite).

L'inconvénient est que, avant de commencer à travailler sur les données, le cloud doit

d'abord opérer homomorphiquement un déchiffrement symétrique, ce qui augmente le *bruit*.

4.1.3 Le bruit en FHE

Sans rentrer dans trop de formalisme, dans tous les chiffrements complètement homomorphes actuels, chaque évaluation d'une fonction sur les données provoque du bruit sur les messages chiffrés. Les opérations XOR provoquent peu de bruit, tandis que les multiplications provoquent beaucoup de bruit. Or, à partir d'un certain niveau de bruit (rapidement atteint), un message chiffré trop bruité donnera le mauvais message clair après déchiffrement.

Ainsi, le paramètre de coût important en FHE est le nombre de multiplications (et en particulier pour certaines générations de FHE, le nombre de multiplications dont dépend chaque bit de sortie, *i.e. profondeur multiplicative*).

Plusieurs approches complémentaires existent pour gérer le bruit : une technique consiste à réduire le bruit en mettant à jour le chiffré avec un chiffré « frais », ce qu'on appelle bootstrapping. Une autre technique consiste à réduire le nombre de multiplications (et/ou la profondeur multiplicative) du chiffrement, pour éviter que le chiffrement n'apporte trop de bruit.

4.1.4 L'outil adapté : le chiffrement à flot

En particulier, dans le cas du chiffrement homomorphe hybride, il faut un chiffrement symétrique dont le déchiffrement provoque peu de bruit, c'est-à-dire avec un faible coût en multiplications.

Plusieurs chiffrements symétriques ont été envisagés pour cette application (des adaptations d'AES [LLN14 ; Che+13 ; DHS14], les chiffrements à bas coût Simon [LN14] et PRINCE [Dor+14] et le chiffrement à flot Trivium [Can+16]), mais étant donné que les chiffrements symétriques n'avaient pas été optimisés dans ce sens au départ, de nouveaux candidats spécifiques pour le FHE ont émergé (en particulier le chiffrement par blocs LowMC [Alb+15] et le chiffrement à flot Kreyvium [Can+16]).

L'avantage des chiffrements par blocs dans ce contexte est que tout chiffré est calculé par le même circuit, utilisant de ce fait le même nombre de multiplications. L'inconvénient est que ce nombre de multiplications est généralement grand.

L'avantage des chiffrements à flot est qu'ils permettent d'obtenir des chiffrés avec très peu de multiplications, mais l'inconvénient est que (jusqu'à FLIP) la profondeur multiplicative augmente pour chaque bit de chiffré consécutif.

4.2 Réduire les coûts au culot : FLIP

4.2.1 Schéma général.

FLIP [Méa+16] a été pensé précisément pour ce cas de figure. Il s'agit d'un chiffrement à flot particulièrement peu coûteux en nombre de multiplications avec un avantage supplémentaire : chaque bit du chiffré dépend du même nombre de multiplications, et donc le bruit est constant (il ne dépend pas de la taille du message).

Ces coûts avantageux sont atteints en utilisant une nouvelle construction dédiée, le *filter permutator* (permutateur filtré), qui utilise une fonction de filtrage F constante et peu coûteuse pour chaque bit du chiffré. De plus, l'état interne de FLIP est constitué de la clef uniquement, et ne change jamais. Ceci est possible car pour chaque bit du chiffré, la fonction de filtrage prend en entrée l'état interne (la clef) permuté de façon aléatoire.

Concrètement, FLIP est constitué de 3 composantes :

— un registre (constant) qui stocke la clef de N bits,

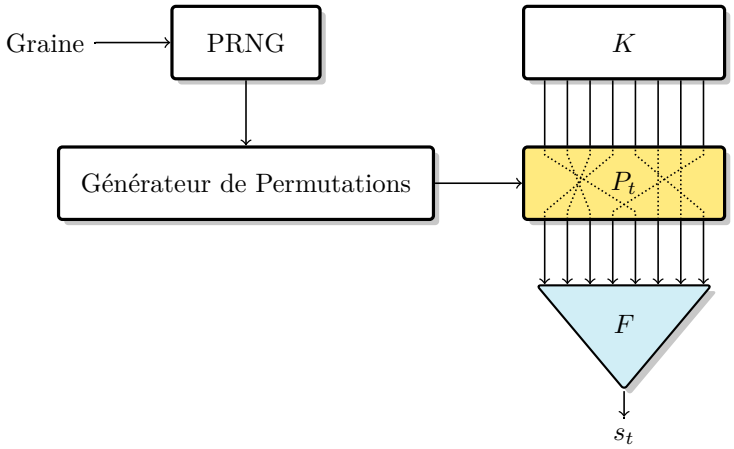


Figure 4.1 – Schéma général du filter permutator utilisé dans FLIP.

— un générateur de permutations public, dérivé d'un un générateur de nombres pseudo-aléatoires (PRNG), qui fournit à chaque itération i une permutation de N bits aléatoire P_i,
— une fonction (booléenne) de filtrage F, qui génère le bit de suite chiffrante z_i à l'itération i.

Une particularité imposée par les auteurs est que la clef doit être *équilibrée* (*i.e.* avoir autant de bits à 0 et à 1). En effet, puisque la clef n'est jamais modifiée, FLIP est très vulnérable à des clefs faibles (en particulier des clefs contenant beaucoup de 0).

Comme c'est l'usage dans les chiffrements à flot, le principe consiste à générer une suite de bits appelée *suite chiffrante*, pseudo-aléatoire, qui est xorée bit à bit pour obtenir le chiffré, suivant l'idée du masque jetable (chiffrement de Vernam).

L'objectif d'un attaquant est alors de trouver des propriétés de la suite chiffrante qui permettent de la distinguer d'une suite aléatoire, voire de prédire les bits de suite chiffrante suivants (et donc de pouvoir déchiffrer tous les bits chiffrés suivants) ou de retrouver la clef (ou l'état interne du chiffrement, qui, dans le cas de FLIP, est réduit à la clef).

Comme habituellement en cryptographie, on part du principe que tous les éléments du chiffrement sont connus de tous (et donc d'un attaquant), hormis la clef. En particulier, un attaquant connaît la permutation P_i à tout temps i.

Bien que ce soit sans importance pour la suite, précisons que le générateur de permutations utilisé est un mélange de Knuth (Knuth shuffle) [Knu69] qui garantit que les permutations ont toutes la même probabilité d'être générées sous réserve que le générateur de nombres aléatoires est bon.

4.2.2 Spécification de la fonction de filtrage F

La fonction de filtrage F a été pensée pour combiner les bonnes propriétés cryptographiques de plusieurs fonctions booléennes, tout en ayant une faible profondeur multiplicative, ce qui, en terme de degré algébrique, veut dire que le degré algébrique est bas.

Formellement, F est définie comme la somme directe de trois fonctions f_1, f_2 et f_3 spécifiées comme suit, avec n et k des entiers positifs et toutes les opérations définies dans $\mathbb{F}_2$.

Définition 4.1 (Fonction de type L)**.** La n-ième fonction de type L, L_n est la fonction

booléenne linéaire à n variables définie par :

$$L_n(x_0, \ldots, x_{n-1}) = \sum_{i=0}^{n-1} x_i \,.$$

Définition 4.2 (Fonction de type Q)**.** La n-ième fonction de type Q, Q_n est la fonction booléenne quadratique à $2n$ variables définie par :

$$L_n(x_0, \ldots, x_{2n-1}) = \sum_{i=0}^{n-1} x_{2i} x_{2i+1} \,.$$

Définition 4.3 (Fonction de type T)**.** La k-ième fonction de type T, T_k est la fonction booléenne triangulaire à $\frac{k(k+1)}{2}$ variables définie par :

$$L_n(x_0, \ldots, x_{n-1}) = \sum_{i=1}^{k} \prod_{j=0}^{i-1} x_{j+\sum_{\ell=0}^{i-1} \ell}, \,.$$

Par exemple, la $3^{\text{ème}}$ fonction de type T est :

$$T_3 = x_0 + x_1 x_2 + x_3 x_4 x_5 \,.$$

Chacune de ces fonctions possède des critères de résistance contre un type d'attaque (ou plus), à savoir non-linéarité, résilience, immunité algébrique,...

La fonction F est définie comme une somme directe de ces trois types de fonctions, paramétrées par les entiers n_1, n_2 et n_3, choisis pour que les propriétés résultantes de F soient bonnes :

— $f_1(x_0, \ldots, x_{n_1-1}) = L_{n_1}$,
— $f_2(x_{n_1}, \ldots, x_{n_1+n_2-1}) = Q_{n_2/2}$,
— $f_3(x_{n_1+n_2}, \ldots, x_{n_1+n_2+n_3-1}) = T_k$, où k est tel que $n_3 = \frac{k(k+1)}{2}$.

F est définie comme la somme directe de f_1, f_2 et f_3, c'est-à-dire :

$$F(x_0, \ldots, x_{n_1+n_2+n_3-1}) = L_{n_1} + Q_{n_2/2} + T_k, \text{ où } n_1 + n_2 + n_3 = N \,.$$

Ainsi, F hérite des bonnes propriétés de f_1, f_2 et f_3 (cf. [Méa+16]).

4.2.3 Analyse de sécurité de Méaux et al.

L'analyse de sécurité des concepteurs de FLIP présentée dans [Méa+16] prend en compte les attaques les plus courantes sur les générateurs filtrés (qui sont en de nombreux points similaires aux permutateurs filtrés), et ont résulté dans les choix de paramètres spécifiés à la table 4.1.

Table 4.1 – Paramètres de sécurité des version préliminaires de FLIP.

FLIP (n_1, n_2, n_3)	Clef $(N = n_1 + n_2 + n_3)$	Sécurité	Degré (k)
FLIP (47, 40, 105)	192	80	14
FLIP (87, 82, 231)	400	128	21

Une étude plus avancée sur des clefs faibles a poussé Méaux *et al.* à imposer le choix d'une clef équilibrée (autant de 0 que de 1).

On notera que la spécification ne donne pas de limite sur le nombre de bits de suite chiffrante auquel un attaquant peut avoir accès.

4.3 Faiblesses de FLIP et idées de l'attaque

Nous commençons par présenter le contexte d'attaque que nous envisageons, puis nous exposons les vulnérabilités de la famille de chiffrements à flot FLIP face aux attaques de type guess-and-determine. Une analyse détaillée de la fonction de filtrage F montre que, combinée à des suppositions ('guess'), on est capable de retrouver les valeurs de bits de la clef ('determine'). Nous détaillons la probabilité qu'une supposition sur les bits de la clef rende la fonction F vulnérable aux attaques algébriques, puis nous dérivons les valeurs théoriques de la complexité de l'attaque.

4.3.1 Scénario et modèle de calcul

Nous nous plaçons dans le scénario le plus usuel pour les attaques sur les chiffrements à flot, à savoir le modèle de clair-connu (l'attaquant peut voir des textes clairs et observer les chiffrés correspondants), ce qui implique que, connaissant des bits de message clair et les chiffrés associés, nous déduisons (par simple xor) les bits de la suite chiffrante z correspondants. Nous utiliserons la notation z_i pour le bit de la suite chiffrante à l'itération numéro i. Notre objectif est alors de retrouver l'état interne du chiffrement à flot, en l'occurrence la clef secrète.

Nous mesurons les performances de notre attaque en utilisant les trois métriques usuelles, c'est-à-dire les complexités en temps, données et mémoire. La complexité en temps (notée C_T) exprime le nombre d'opérations nécessaires à l'attaquant pour appliquer son attaque. Nous compterons les opérations comme dans la spécification de FLIP ([Méa+16]), *i.e.* nous comptons le nombre d'opérations de base. La complexité en données (C_D) correspond au nombre de bits de suite chiffrante requis, et la complexité en mémoire (C_M) mesure la quantité de mémoire (en bits) nécessaire pour appliquer l'attaque.

4.3.2 Vulnérabilités de la famille FLIP face aux attaques de type guess-and-determine

Notre attaque utilise une variante de la technique baptisée guess-and-determine. Cette approche, qui semble avoir été nommée initialement dans [EJ00 ; HR00], a été maintes fois utilisée pour attaquer des chiffrements à flot, à commencer par ceux proposés au projet NESSIE [2]. L'idée est de commencer par faire une hypothèse sur la valeur de certains bits de l'état interne ou la clef ('Guess') puis, en observant les bits de suite chiffrante associés, de déterminer la valeur de bits inconnus ('Determine'). Généralement, l'attaque est complétée par des techniques algébriques.

Deux aspects de la famille de chiffrements FLIP semblent indiquer qu'une attaque de type guess-and-determine serait efficace : d'une part son état interne fixé et d'autre part la définition de sa fonction de filtrage F.

4.3.2.1 Vulnérabilité due à l'état interne constant

Plus précisément, le fait que le registre qui stocke l'état interne ne soit jamais mis à jour implique qu'une supposition sur un bit de la clef/état interne à n'importe quelle itération nous donne une information à toutes les autres itérations. C'est une situation très différente des chiffrements à flot usuels dont l'état interne est modifié à chaque itération, ce qui induit qu'une information d'un bit disparaît très rapidement après quelques tours (en avant ou en arrière).

2. https://www.cosic.esat.kuleuven.be/nessie/

4.3.2.2 Vulnérabilité due à la forme de la fonction de filtrage

La deuxième caractéristique exploitable dans FLIP est sa fonction de filtrage, qui contient très peu de monômes de haut degré.

Comme expliqué en section 4.2.2, la fonction de filtrage F est la somme directe de trois fonctions f_1, f_2 et f_3 qui sont respectivement de type L, Q et T. Cette définition implique que tous les monômes de degré supérieur ou égal à 3 sont contenus dans f_3, qui est donnée par la formule suivante :

$$\begin{aligned} f_3(x_{n_1+n_2}, \cdots, x_{n_1+n_2+n_3-1}) &= T_k(x_{n_1+n_2}, \cdots, x_{n_1+n_2+n_3-1}) \\ &= \sum_{i=1}^{k} \prod_{j=n_1+n_2}^{n_1+n_2+i-1} x_{j+\sum_{\ell=0}^{i-1}\ell} \end{aligned}$$

où k est le degré algébrique de f_3 et est tel que $n_3 = \frac{k(k+1)}{2}$.

Cette expression contient $k-2$ monômes de degré supérieur ou égal à 3, qui sont les seuls dans F, impliquant un total de $n_3 - 3$ variables. D'après la contrainte de profondeur multiplicative, k doit être petit [3] ce qui veut dire que F contient très peu de monômes de degré supérieur ou égal à 3, et que donc il est aisé d'annuler tous ces monômes (et ainsi de se retrouver avec une fonction quadratique). C'est le sujet du paragraphe 4.3.2.3.

L'idée essentielle de notre attaque est de remarquer que puisque F ne contient que peu de monômes de degré supérieur ou égal à 3, mais beaucoup de bits de clefs égaux à 0 (la moitié), il y a de fortes chances que les monômes de haut degré soient annulés. Lorsqu'un tel cas se produit, les bits de suite chiffrante peuvent être vus comme des expressions de degré 2 en les bits non-nuls de la clef.

Notre attaque utilise cette spécificité en opérant une variante de la technique de guess-and-determine : plutôt que de faire une supposition sur la valeur des bits de la clef, nous supposons les indices de certains bits non-nuls de la clef [4]. Nous en déduisons les numéros d'itération où les bits de la suite chiffrante sont des expressions de degré faible en les bits non-nuls de la clef que nous utilisons pour construire un système d'équations quadratiques. Finalement, en utilisant des techniques de linéarisation, nous résolvons le système quadratique et retrouvons les valeurs des bits de la clef.

4.3.2.3 Probabilité d'annuler les monômes de haut degré de F sachant que ℓ variables d'entrée sont nulles.

Afin de déterminer la faisabilité du procédé, il nous faut évaluer la probabilité que, étant données exactement ℓ positions de bits non-nuls dans la clef K, l'expression du bit de suite chiffrante z_i soit de degré inférieur ou égal à 2 en les bits de clef restants [5].

Cette probabilité est directement liée à la quantité de données nécessaires pour obtenir une attaque puisqu'elle détermine le nombre de bits de suite chiffrante qu'un attaquant doit parcourir pour qu'assez de ces bits soient exploitables pour construire le système d'équations quadratiques.

De la discussion précédente sur les spécifications de FLIP, nous savons qu'il y a exactement $k-2$ monômes disjoints de degré supérieur ou égal à 3 dans l'expression de $z_i = F(P_i(k_0, k_1, \cdots k_{N-1}))$. De ce fait, si l'attaquante n'a accès qu'à $\ell < k-2$ positions nulles, elle ne pourra pas déterminer de moments exploitables, ce qui force que $\ell \geqslant k-2$, *i.e.* au minimum un bit à zéro positionné dans chaque monôme de haut degré.

3. Concrètement, on rappelle que les deux instances de FLIP de [Méa+16] utilisent $k = 14$ et $k = 21$ respectivement pour une sécurité de 80 et 128 bits.

4. Comme précisé dans la définition de FLIP, nous savons qu'il y en a exactement $\frac{N}{2}$.

5. Nous parlerons alors d'*équation exploitable* ou de *moment exploitable*.

Cas 1 : si $\ell = k-2$. La première possibilité est de choisir un nombre de positions nulles égal au nombre de monômes de haut degré que nous voulons annuler, *i.e.* $\ell = k-2$. Dans ce cas, il faut exactement un bit nul dans chaque monôme de haut degré : par exemple si nous nous focalisons sur un monôme spécifique de degré d : $x_0 x_1 \cdots x_{d-1}$, c'est équivalent à choisir laquelle des variables x_i, $0 \leqslant i \leqslant d-1$ est nulle, ce qui offre d possibilités. Ainsi, nous pouvons énumérer l'ensemble des configurations valides, ce qui correspond à choisir un indice dans chaque monôme. Puisqu'il y a 3 choix possibles dans le monôme de degré 3, 4 possibilités pour celui de degré 4 et ainsi de suite jusqu'au monôme de degré k, il y a un total de $3 \times 4 \times 5 \cdots \times k = k!/2$ configurations valides. Pour obtenir la probabilité, il faut comparer cette quantité au nombre total de possibilités de choix des positions nulles, qui est $\binom{N}{\ell}$, et nous avons donc :

$$\mathbb{P}_{\ell=k-2} = \frac{k!/2}{\binom{N}{\ell}}.$$

Cas général : si $\ell \geqslant k-2$. Pour augmenter la probabilité qu'un moment soit exploitable, l'attaquante peut deviner plus de positions de bits nuls et choisir $\ell \geqslant k-2$. Une première façon de calculer la probabilité d'annuler alors tous les monômes de degré supérieur ou égal à 3 est :

$$\mathbb{P}_\ell = \frac{\sum_{i_1+i_2+\cdots+i_{k-2} \leqslant \ell} \binom{3}{i_1}\binom{4}{i_2}\cdots\binom{k}{i_{k-2}}\binom{N-m}{\ell-I}}{\binom{N}{\ell}}$$

où m est le nombre de variables qui apparaissent dans les monômes de degré supérieur ou égal à 3 et $I = i_1 + i_2 + \cdots + i_{k-2}$ et $i_i \geqslant 0$.

Démonstration. Supposons données ℓ positions de bits nuls dans K. Nous voulons obtenir la probabilité qu'une permutation aléatoire P_i mélange les bits de clef de telle façon que l'évaluation de F ne contienne aucun monôme de degré supérieur ou égal à 3. Comme précédemment, nous comptons le nombre de configurations valides par rapport au nombre total de permutations.

Il s'agit alors de lister toutes les manières de positionner au moins un bit nul dans chaque monôme : nous fixons i_1 bits nuls dans le monôme de degré 3, i_2 dans le monôme de degré 4, et ainsi jusqu'à i_{k-2} bits nuls dans le monôme de plus haut degré (k). Si nous notons $I = i_1 + i_2 + \cdots + i_{k-2}$ le nombre de bits positionnés ainsi, il nous reste $\ell - I$ positions de bits nuls à répartir entre les $N - m$ autres monômes. Pour obtenir la probabilité, il nous faut diviser cette quantité par le nombre de façons de positionner ℓ bits parmi N positions de bits. □

Une autre approche qui amène à la même probabilité est de calculer le nombre de configurations qui n'annulent pas les monômes de degré supérieur ou égal à 3, qui est la probabilité complémentaire de celle que nous cherchons. L'avantage est que cette probabilité complémentaire peut s'exprimer aisément par un principe d'inclusion-exclusion. Notons A_J l'événement que notre supposition n'annule pas les monômes de degré inclus dans l'ensemble J, *i.e.* :

$$A_J \text{ est l'événement : } \{\forall j \in J,\, M_j \neq 0\}$$

où M_j est l'unique monôme de degré j dans T_k.

$\mathbb{P}(A_J)$ est la probabilité de fixer les ℓ positions de bits parmi les monômes dont le degré n'est pas dans J et est donc égale à :

$$\mathbb{P}(A_J) = \frac{\binom{N - \sum_{j \notin J} j}{\ell}}{\binom{N}{\ell}}$$

Nous pouvons maintenant exprimer la probabilité qu'une supposition donne un polynôme de degré supérieur ou égal à 3 par :

$$\mathbb{P}(\bigcup_{d\in\{3,\cdots,k\}} A_{\{d\}}) = \sum_{s=1}^{k-2}\left((-1)^s \sum_{\substack{J\subseteq\{3,\cdots,k\}\\|J|=s}} \mathbb{P}(A_J)\right)$$

qui peut se réécrire en :

$$\mathbb{P}(\bigcup_{d\in\{3,\cdots,k\}} A_{\{d\}}) = \frac{\sum_{s=1}^{k-2}\left((-1)^s \sum_{\substack{J\subseteq\{3,\cdots,k\}\\|J|=s}} \binom{N-\sum_{j\notin J} j}{\ell}\right)}{\binom{N}{\ell}}.$$

Un avantage de cette approche est que $\frac{\binom{N-a}{\ell}}{\binom{N}{\ell}} = \frac{(N-a)!}{(N-a-\ell)!}\frac{(N-\ell)!}{N!}$. Ainsi, nous pouvons calculer l'expression aisément par 2ℓ divisions de nombres plus petits que N et ℓ produits de nombres plus petits que 1, ce qui à la fois est efficace et évite de manipuler des (très) grands nombres.

Nous en déduisons l'expression de la probabilité que nous recherchons :

$$\mathbb{P}_\ell = \mathbb{P}(\bigcap_{d\in\{3,\cdots,k\}} \overline{A_{\{d\}}}) = 1 - \mathbb{P}(\bigcup_{d\in\{3,\cdots,k\}} A_{\{d\}}).$$

L'évaluation de ces formules donne les résultats reportés aux tables 4.2 et 4.3 et nous verrons dans la section suivante que ces résultats sont suffisants pour obtenir une attaque. Par exemple, si nous attaquons la plus petite version [6] de FLIP en utilisant un nombre de suppositions minimal (*i.e.* $\ell = 12$) nous obtenons une probabilité d'avoir une équation exploitable de $\mathbb{P}_{\ell=12} = 2^{-26.335}$. Pour l'autre version [7] et un nombre de suppositions minimal nous avons $\mathbb{P}_{\ell=19} = 2^{-42.382}$.

4.4 Notre attaque

4.4.1 Description

Mise en place. Puisque nous nous plaçons dans un scénario de clair-connu, nous supposons qu'on nous donne C_D bits de suite chiffrante que nous noterons z_i, $i = 0, \cdots, C_D - 1$. De plus, les permutations associées P_i étant publiques, nous connaissons les expressions des bits de suite chiffrante en tant que fonctions des bits de clef inconnus $k_0, \cdots, k_{N-1}$:

$$z_i = F(P_i(k_0, k_1, \cdots, k_{N-1})) \quad \forall i \geqslant 0$$

Notre attaque tire avantage des deux vulnérabilités détaillées dans la section précédente afin de ramener le problème de la récupération de la clef à la résolution d'un système linéarisé.

Étape 1 : supposition initiale. La première étape consiste à faire une hypothèse sur les positions de ℓ bits nuls de la clef, où $\ell \geqslant k - 2$. Supposer ces bits nuls nous donne une expression simplifiée de z_i en seulement $N - \ell$ inconnues. Puisque la clef K est équilibrée, la probabilité que notre supposition soit correcte est [8] :

6. FLIP (47,40,105)
7. FLIP (87,82,231)
8. Cette probabilité est légèrement plus faible que pour une clef aléatoire ($2^{-\ell}$), mais l'avantage est que tant que nous supposons $\ell \leqslant \frac{N}{2}$, nous sommes certains qu'au moins une supposition sera correcte alors que nos suppositions pourraient échouer pour une clef aléatoire qui n'a pas assez de bits nuls.

$$\mathbb{P}_{rg} = \frac{\binom{\frac{N}{2}}{\ell}}{\binom{N}{\ell}}.$$

Étape 2 : extraction d'équations de bas degré. L'objectif de la deuxième étape est d'amasser des équations de bas degré en les bits inconnus de la clef. Pour ce faire, nous inspectons les expressions des z_i disponibles et nous prenons toutes les équations pour lesquelles les bits nuls de la clef annulent les monômes de degré supérieur ou égal à 3. Comme détaillé dans la section précédente, cet événement a une probabilité $\mathbb{P}_\ell$.

Étape 3 : résolution du système. Il existe de nombreuses façons de résoudre des systèmes d'équations quadratiques. Dans notre cas, nous nous contenterons d'une approche simple qui consiste à utiliser des techniques de linéarisation [9]. La linéarisation consiste à convertir le système en un système linéaire par l'introduction de nouvelles variables pour chaque monôme non-linéaire qui apparaît. Dans notre cas spécifique, les seules expressions non-linéaires qui interviennent sont des monômes de degré 2. Comme F prend en entrée N variables mais que nous en avons deviné ℓ, il nous reste $N - \ell$ variables inconnues, qui dans le pire cas forment $\binom{N-\ell}{2}$ monômes de degré 2. Ceci implique qu'une fois linéarisé, notre système contiendra au plus

$$v_\ell = N - \ell + \binom{N-\ell}{2}$$

variables.

En supposant que les équations sont aléatoires, le nombre d'équations nécessaires pour obtenir une solution unique (ou pour obtenir une contradiction qui indiquerait que notre supposition des positions nulles est fausse) est de l'ordre du nombre d'inconnues [10]. Ceci implique que le nombre de bits de suite chiffrante nécessaires à l'attaquante doit être le produit du nombre de variables et de l'inverse de la probabilité qu'un bit de suite chiffrante z_i soit exploitable :

$$C_D = v_\ell \times \frac{1}{\mathbb{P}_\ell}$$

La complexité en temps est déterminée par le temps nécessaire à la résolution du système [11] multiplié par le nombre de fois que nous devons répéter la supposition de ℓ positions de bits nuls avant d'en trouver une correcte :

$$C_T = v_\ell^3 \times \frac{1}{\mathbb{P}_{rg}}$$

La complexité finale en mémoire est dominée par la mémoire nécessaire pour stocker le système, donc de l'ordre de :

$$C_M = v_\ell^2$$

Les tables 4.2 et 4.3 donnent les compromis possibles entre les complexités en temps et en données pour les deux versions de FLIP. Comme nous pouvons le voir, augmenter le nombre de suppositions initiales ℓ permet de réduire le nombre de données nécessaires pour faire l'attaque en sacrifiant du temps de calcul.

9. Ce n'est pas nécessairement l'approche la plus efficace, mais sans optimiser plus avant cette étape, l'attaque est déjà performante.

10. Ce qui est confirmé par nos expériences détaillées en section 4.6.

11. Qui est v_ℓ^3 pour une simple élimination gaussienne ou $v_\ell^{2.8}$ avec l'algorithme de Strassen. Nous utilisons le premier choix par souci de simplicité.

Table 4.2 – Logarithme en base 2 des complexités des attaques en fonction du nombre de suppositions initiales (ℓ) pour l'instance $FLIP(47, 40, 105)$.

ℓ	$\mathbb{P}_\ell$	v_ℓ	$\mathbb{P}_{rg}$	C_D	C_T	C_M
12	-26.335	13.992	-12.528	40.326	54.503	27.983
13	-23.049	13.976	-13.627	37.025	55.554	27.951
14	-20.653	13.960	-14.736	34.613	56.615	27.919
15	-18.738	13.943	-15.854	32.682	57.684	27.887
16	-17.141	13.927	-16.982	31.069	58.763	27.854
17	-15.775	13.911	-18.120	29.686	59.852	27.821
18	-14.585	13.894	-19.267	28.480	60.950	27.788
19	-13.536	13.878	-20.425	27.414	62.057	27.755
20	-12.601	13.861	-21.592	26.462	63.175	27.722
21	-11.762	13.844	-22.771	25.606	64.303	27.688
22	-11.004	13.827	-23.960	24.831	65.442	27.654
23	-10.315	13.810	-25.160	24.125	66.591	27.621
24	-9.686	13.793	-26.371	23.479	67.750	27.586
25	-9.110	13.776	-27.593	22.886	68.921	27.552
26	-8.580	13.759	-28.827	22.339	70.103	27.517
27	-8.092	13.741	-30.073	21.833	71.297	27.483
28	-7.640	13.724	-31.331	21.364	72.502	27.448
29	-7.221	13.706	-32.601	20.927	73.720	27.413
30	-6.832	13.689	-33.883	20.520	74.949	27.377
31	-6.469	13.671	-35.179	20.140	76.191	27.342
32	-6.131	13.653	-36.487	19.784	77.446	27.306
33	-5.816	13.635	-37.809	19.450	78.714	27.270
34	**-5.520**	**13.617**	**-39.145**	**19.137**	**79.995**	**27.233**
> 34	...	...	...	...	> 80	...
35	-5.243	13.598	-40.495	18.842	81.290	27.197

Table 4.3 – Logarithme en base 2 des complexités de l'attaque en fonction du nombre initial de suppositions (ℓ) pour l'instance $FLIP(87, 82, 231)$.

ℓ	$\mathbb{P}_\ell$	v_ℓ	$\mathbb{P}_{rg}$	C_D	C_T	C_M
19	-42.382	16.151	-19.647	58.533	68.100	32.302
20	-38.522	16.144	-20.721	54.666	69.151	32.287
21	-35.589	16.136	-21.799	51.725	70.206	32.272
22	-33.169	16.128	-22.881	49.298	71.266	32.257
23	-31.097	16.121	-23.967	47.218	72.329	32.241
24	-29.282	16.113	-25.058	45.395	73.397	32.226
25	-27.667	16.105	-26.153	43.772	74.469	32.211
26	-26.214	16.098	-27.253	42.311	75.546	32.195
27	-24.895	16.090	-28.357	40.985	76.627	32.180
28	-23.691	16.082	-29.465	39.773	77.712	32.164
29	-22.584	16.074	-30.578	38.658	78.802	32.149
30	-21.562	16.067	-31.696	37.629	79.896	32.133
31	-20.615	16.059	-32.818	36.674	80.994	32.118
32	-19.734	16.051	-33.944	35.785	82.097	32.102
33	-18.912	16.043	-35.075	34.955	83.205	32.086
34	-18.142	16.035	-36.211	34.178	84.317	32.071
35	-17.421	16.027	-37.352	33.448	85.434	32.055
36	-16.743	16.020	-38.497	32.762	86.556	32.039
37	-16.104	16.012	-39.648	32.116	87.683	32.023
38	-15.502	16.004	-40.803	31.505	88.814	32.007
39	-14.932	15.996	-41.963	30.928	89.950	31.991
40	-14.393	15.988	-43.128	30.381	91.091	31.975
41	-13.883	15.980	-44.298	29.862	92.237	31.959
42	-13.398	15.972	-45.473	29.370	93.388	31.943
43	-12.937	15.964	-46.653	28.901	94.543	31.927
44	-12.499	15.956	-47.838	28.455	95.704	31.911
45	-12.082	15.947	-49.028	28.029	96.870	31.895
46	-11.684	15.939	-50.224	27.624	98.042	31.879
47	-11.305	15.931	-51.425	27.236	99.218	31.862
48	-10.942	15.923	-52.631	26.865	100.400	31.846
49	-10.596	15.915	-53.842	26.511	101.586	31.830

Table 4.4 – Logarithme en base 2 des complexités de l'attaque en fonction du nombre initial de suppositions (ℓ) pour l'instance $FLIP(87, 82, 231)$.

ℓ	$\mathbb{P}_\ell$	v_ℓ	$\mathbb{P}_{rg}$	C_D	C_T	C_M
50	-10.265	15.907	-55.059	26.171	102.779	31.813
51	-9.948	15.898	-56.282	25.846	103.976	31.797
52	-9.644	15.890	-57.509	25.534	105.180	31.780
53	-9.353	15.882	-58.743	25.235	106.388	31.763
54	-9.074	15.873	-59.982	24.947	107.602	31.747
55	-8.806	15.865	-61.227	24.671	108.822	31.730
56	-8.548	15.857	-62.477	24.405	110.048	31.713
57	-8.301	15.848	-63.734	24.149	111.279	31.697
58	-8.063	15.840	-64.996	23.903	112.516	31.680
59	-7.835	15.831	-66.264	23.666	113.758	31.663
60	-7.614	15.823	-67.538	23.437	115.007	31.646
61	-7.402	15.815	-68.818	23.217	116.262	31.629
62	-7.198	15.806	-70.104	23.004	117.522	31.612
63	-7.001	15.797	-71.397	22.799	118.789	31.595
64	-6.812	15.789	-72.695	22.601	120.062	31.578
65	-6.629	15.780	-74.000	22.409	121.341	31.561
66	-6.452	15.772	-75.311	22.224	122.627	31.543
67	-6.281	15.763	-76.629	22.044	123.918	31.526
68	-6.116	15.754	-77.953	21.871	125.216	31.509
69	-5.957	15.746	-79.284	21.703	126.521	31.491
70	**-5.803**	**15.737**	**-80.621**	**21.540**	**127.832**	**31.474**
> 70	...	...	...	...	> 128	...
71	-5.655	15.728	-81.965	21.383	129.150	31.457

4.4.2 Discussion et possibilités d'améliorations

Réduction de la complexité en données. La complexité en données peut être améliorée si, plutôt que de choisir des suppositions aléatoires, l'attaquante choisit les suppositions par rapport aux permutations observées. Puisque la graine du PRNG est publique, à n'importe quel moment, l'attaquante connaît toutes les permutations à venir et peut donc choisir une supposition sur les positions nulles de la clef qui annule la partie triangulaire pour un grand nombre des permutations à venir.

Possibilité de précalculs. La majeure partie du coût de calcul de l'attaque repose sur la résolution d'un système linéaire. Remarquons que ce système linéaire dépend uniquement de la permutation et de la supposition à un moment fixé, qui sont des données connues de l'attaquante, qui peut donc calculer cette inversion de système pour plusieurs suppositions sans connaître la suite chiffrante. Lorsqu'elle reçoit la suite chiffrante, elle l'introduit dans ses précalculs pour obtenir les résultats. L'inconvénient de cette technique est le surcoût en mémoire.

Indépendance de la graine. Notre attaque a la propriété de ne pas être affectée par une réinitialisation du système, dans le sens où un changement de graine du PRNG au milieu de l'attaque ne force pas l'attaquante à recommencer son attaque : elle peut combiner les équations obtenues précédemment avec celles obtenues avec la nouvelle graine.

Sécurité. Le niveau de sécurité de la famille de chiffrements à flot FLIP est au mieux proportionnel à $\sqrt{N}$ bits, où N est la taille de la clef.

Démonstration. La complexité en temps de l'attaque est

$$C_T = v_\ell^3 \times \frac{1}{\mathbb{P}_{rg}} .$$

Comme $\ell \ll N$, on peut dire que $\mathbb{P}_{rg}$ est de l'ordre de $2^{-\ell}$. De plus, puisque $v_\ell = N - \ell + \binom{N-\ell}{2}$, nous pouvons approximer C_T par

$$C_T \sim N^6 \times 2^\ell .$$

De surcroît, le nombre de suppositions qu'il faut pour l'attaque est le nombre de monômes de degré supérieur ou égal à 3 dans $T_{n,3}$. Ainsi $n_3 = (\ell+2)(\ell+3)/2$, donc $\ell \sim \sqrt{n_3}$, d'où :

$$\log C_T \sim \alpha\sqrt{N} .$$

□

La figure 4.2 représente l'évolution de la complexité en temps de notre attaque en fonction de la taille de clef lorsque nous considérons des instances de FLIP de la forme $FLIP(n_1, n_2, n_3)$ où $N = n_1 + n_2 + n_3 = 2n_3$ (ce qui est similaire aux paramètres proposés dans [Méa+16]). ℓ est choisi comme le nombre de suppositions minimal pour obtenir une attaque, *i.e.* $\ell = k - 2$.

Tentative d'annulation de la partie quadratique. Notre attaque consiste à supposer la position de bits nuls de la clef pour annuler les monômes de degré supérieur ou égal à 3 de la fonction de filtrage. Une autre possibilité serait d'annuler les monômes de degré 2 afin de réduire la résistance de FLIP contre les attaques par corrélation. Nous avons considéré cette option, mais nos études ont montré que la complexité d'une telle attaque serait trop grande. Cependant, pour une version modifiée de FLIP (et en particulier la

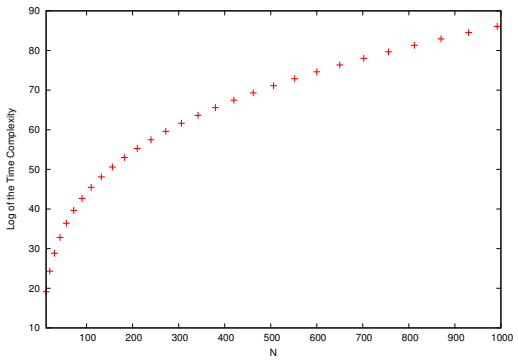


Figure 4.2 – Évolution de la complexité en temps en fonction de la taille de clef N.

version publiée [Méa+16]), annuler la partie quadratique plutôt que la partie de degré supérieur ou égal à 3 est potentiellement plus avantageux et ne doit pas être négligé. Nous avons aussi envisagé d'annuler à la fois la partie quadratique et la partie de degré supérieur ou égal à 3, ce qui ne laisserait que des relations linéaires, mais la complexité en données d'une telle attaque la rend infaisable en pratique.

4.5 Description de l'algorithme

La partie principale de notre attaque est une résolution de système linéaire sur $\mathbb{F}_2$. Si la résolution détecte une contradiction, nous déduisons que notre supposition sur les bits de la clef est fausse, et recommençons avec une nouvelle supposition. Dans le cas contraire, la supposition était correcte et résoudre le système donne la clef. L'intuition veut qu'on n'aie pas toujours besoin d'un système de rang plein pour détecter une contradiction. Nous pouvons donc améliorer l'attaque en traitant les équations à la volée, plutôt que d'attendre d'avoir un système de rang plein.

Une description en pseudocode de l'attaque qui utilise cette amélioration est donnée dans l'algorithme 2. Dans cet algorithme, une équation est représentée par un mot de $v_\ell + 1$ bits contenant un 1 là où une variable est présente et un 0 sinon. Le bit de poids faible de cette représentation contient la valeur du bit de la suite chiffrante de l'équation. Nous mémorisons aussi si l'équation i est présente dans le système dans un vecteur $Existe$. Si $Existe[i] = 1$, c'est que l'équation i est dans le système.

4.6 La théorie face à la pratique : test sur une version jouet

Pour vérifier et étayer nos résultats, nous avons implémenté l'attaque sur une version jouet du chiffrement. Nous avons réduit la taille de clef à $N = 64$ bits et avons adapté les valeurs des paramètres en conséquence à $n_1 = 14$, $n_2 = 14$ et $n_3 = 36$ (les proportions entre les tailles des paramètres sont conservées). La fonction de filtrage F est de degré algébrique 8 est définie comme suit :

$$F(x_0, \cdots x_{63}) = f_1(x_0, \cdots x_{13}) + f_2(x_{14}, \cdots x_{27}) + f_3(x_{28} \cdots x_{63})$$

Algorithm 2 FLIP Algorithme de recouvrement de clef

1: **function** RECOUVREMENT DE CLEF
Entrée : Suite chiffrante, graine du PRNG.
2: **Sortie :** Clef.

3: SYSTEME ← Vecteur de v_ℓ mots nuls
4: Existe ← Vecteur de v_ℓ bits nuls
5: ClefPasEncoreTrouvée ← VRAI
6: **while** ClefPasEncoreTrouvée **do**
7: G ← NouvelleSuppositionAléatoire
8: PasDeContradiction ← FAUX
9: $N_{eq} \leftarrow 0$
10: **while** PasDeContradiction ET $N_{eq} \leqslant v_\ell$ **do**
11: E ← NouvelleEquation
12: NouvelIndice ← -1
13: i ← MSB(E)
14: **while** i $\leqslant v_\ell$ **do**
15: **if** Existe[i] **then**
16: E ← E $\oplus$ SYSTEME[i]
17: i ← MSB(E)
18: **else**
19: **if** NouvelIndice $= -1$ **then**
20: NouvelIndice ← i
21: i ← Indice du prochain bit de valeur 1 à partir de l'indice i + 1 en allant vers le LSB
22: **for** j = 1 à NouvelIndice **do**
23: **if** Existe[i] ET SYSTEME[i][NouvelIndice] = 1 **then**
24: SYSTEME[i] ← SYSTEME[i] $\oplus$ E
25: **if** E = 1 **then**
26: PasDeContradiction ← FAUX
27: **else**
28: **if** E $\neq 0$ **then**
29: SYSTEM[NouvelIndice] ← E
30: $N_{eq} \leftarrow N_{eq} + 1$
31: Si E = 0, l'équation est linéairement dépendante des premières mais n'apporte pas de contradiction, on n'augmente pas N_{eq}
32: **if** PasDeContradiction **then**
33: Obtenir x_i et $x_i x_j$ et Tester s'il y a contradiction
34: **if** Il n'y a pas de contradiction **then**
35: ClefPasEncoreTrouvee ← FAUX
36: Clef ← $(x_i)_{1 \leqslant i \leqslant n}$
return Clef

où

$$\begin{aligned} f_1(x_0,\cdots,x_{13}) &= L_{14}(x_0,\cdots,x_{13}) &&= x_0 + x_1 + \cdots + x_{13} \\ f_2(x_{14},\cdots,x_{27}) &= Q_7(x_{14},\cdots,x_{27}) &&= x_{14}x_{15} + x_{16}x_{17} + \cdots + x_{26}x_{27} \\ f_3(x_{28},\cdots,x_{63}) &= T_8(x_{28},\cdots,x_{63}) &&= x_{28} + x_{29}x_{30} + x_{31}x_{32}x_{33} + \cdots + x_{56}x_{57}\cdots x_{63} \end{aligned}$$

D'après nos analyses, la complexité théorique de l'attaque correspond à la table 4.6 : par exemple, si nous décidons de faire une supposition sur $\ell = 8$ indices à zéro, la probabilité que notre supposition soit correcte est

$$\mathbb{P}_{rg} = 2^{-8.717}\,.$$

La probabilité que la permutation soit exploitable vaut :

$$\mathbb{P}_{\ell} = 2^{-7.814}$$

et le système linéarisé dépend de $v_\ell = 1596$ variables. Nous prévoyons qu'il faille $C_D = 2^{18.454}$ bits pour faire notre attaque et que le temps requis soit de $C_T = 2^{40.638}$ opérations élémentaires.

Nous avons implémenté notre propre version de ce chiffrement-jouet sur lequel nous avons effectué notre attaque avec $\ell = 8$ suppositions. Les statistiques obtenues sont données à la table 4.5.

Bien que les équations aient une forme très spécifique, nous notons qu'elles se comportent comme des équations aléatoires, au sens suivant : la première équation linéairement dépendante des précédentes est obtenue après avoir généré 1590 équations, ce qui correspond parfaitement à la théorie dans le cas des équations aléatoires [LN83]. Cependant, traiter les équations à la volée nous permet d'éliminer directement les équations qui sont linéairement dépendantes des autres. De cette façon, nous pouvons arrêter de collecter des équations dès que nous avons autant d'équations que de variables [12].

Table 4.5 – Comparaison des résultats expérimentaux et théoriques : attaque sur une version jouet avec $FLIP(14, 14, 36)$ et une supposition sur $\ell = 8$ positions de bits nuls (en moyenne sur 1000, lancé sur un CPU Intel(R) Xeon(R) W3670 à 3.20GHz (12MB de cache), et avec 8GB de RAM)

	Suppositions	Données	Ratio exploité	Op. Élémentaires	Temps (sec)
Pratique	437.1	$2^{18.455}$	$2^{-7.813}$	$2^{38.588}$	280.93
Théorie	$\mathbb{P}_{rg}^{-1} = 420.8$	$C_D = 2^{18.454}$	$\mathbb{P}_{\ell} = 2^{-7.814}$	$C_T = 2^{40.638}$	∅

12. Les expériences montrent que nous éliminons à peu près 500 équations avant d'en avoir 1596 indépendantes.

Table 4.6 – Logarithme en base 2 des complexités de l'attaque en fonction du nombre initial de suppositions (ℓ) pour la version jouet $FLIP(14, 14, 36)$.

ℓ	$\mathbb{P}_\ell$	v_ℓ	$\mathbb{P}_{rg}$	C_D	C_T	C_M
6	-11.861	10.741	-6.370	22.601	38.592	21.481
7	-9.436	10.691	-7.528	20.126	39.601	21.382
8	-7.814	10.640	-8.717	18.454	40.638	21.280
9	-6.602	10.589	-9.939	17.191	41.706	21.177
10	-5.649	10.536	-11.197	16.185	42.806	21.072
11	-4.876	10.483	-12.493	15.359	43.941	20.966
12	-4.235	10.428	-13.828	14.663	45.113	20.857
13	-3.696	10.373	-15.207	14.069	46.325	20.746
14	-3.237	10.316	-16.631	13.553	47.580	20.633
15	-2.843	10.259	-18.105	13.102	48.881	20.517
16	-2.503	10.200	-19.632	12.703	50.231	20.399
17	-2.208	10.140	-21.217	12.347	51.636	20.279
18	-1.950	10.078	-22.865	12.028	53.100	20.156
19	-1.723	10.015	-24.581	11.739	54.628	20.031
20	-1.524	9.951	-26.373	11.476	56.227	19.903
21	-1.349	9.886	-28.247	11.235	57.904	19.771
22	-1.194	9.819	-30.214	11.013	59.670	19.637
23	-1.057	9.750	-32.284	10.807	61.534	19.500
24	**-0.935**	**9.679**	**-34.472**	**10.615**	**63.510**	**19.359**
> 24	...	...	...	...	> 64	...
25	-0.827	9.607	-36.794	10.435	65.616	19.215

4.7 On a sauvé FLIP : une version résistante

Nous avons communiqué nos analyses aux auteurs de FLIP, qui ont pu mettre à jour leur chiffrement avant la publication. Ainsi, la version publiée est résistante à ces attaques. Après des discussions avec les auteurs, voici les paramètres de FLIP qui ont finalement été adoptés :

— Modification de la fonction de filtrage F :

$$F = L(x_0, \ldots, x_{n_1-1}) + Q(x_{n_1}, \ldots, x_{n_1+n_2-1}) +^{nb} \Delta^k(x_{n_1+n_2}, \ldots, x_{N-1})$$

avec ${}^{nb}\Delta^k = \sum_{i=1}^{nb} T_k$.
Ceci apporte un plus grand nombre de monômes de degré supérieur ou égal à 3.

— Des clefs drastiquement plus grandes.

Une analyse rapide permet d'obtenir les complexités d'attaque suivantes pour les deux versions publiées de FLIP :

$FLIP(n_1, n_2, {}^{nb}\Delta^k)$	Clef (N)	Sécurité	Temps	Données	Mémoire
$FLIP(42, 128, {}^{8}\Delta^9)$	**530**	80	2^{111}	$2^{130.8}$	$2^{33.56}(+2^{130.8})$
$FLIP(82, 224, {}^{8}\Delta^{16})$	**1394**	128	$2^{169.1}$	$2^{505.67}$	$2^{39.4}(+2^{505.67})$

Nous constatons donc que, telle quelle, notre attaque ne fonctionne plus sur les nouvelles versions (ou versions publiées) de FLIP.

4.8 Conclusion de la cryptanalyse de FLIP

Dans ces travaux, nous avons montré l'existence d'une attaque (coûtant 2^{54} et 2^{68} opérations respectivement pour les 2 versions, donc à la limite d'être pratique) sur la famille de chiffrements à flot FLIP. De plus, nous proposons divers compromis et améliorations qui peuvent encore réduire la complexité de l'attaque.

En l'occurrence, FLIP avait été conçu avec une structure très différente des chiffrements à flot usuels afin de coûter peu cher pour le FHE, et nous démontrons que cette approche n'est pas très viable en général (puisque la complexité de l'attaque croît avec la racine carrée de la taille de clef). Toujours est-il que, dans le cas particulier du FHE, même avec une clef grande, FLIP reste l'une des meilleures solutions connues à ce jour.

Découvrir cette attaque a permis de s'en prémunir, mais il y a deux leçons à tirer de cette expérience :

— lorsqu'on propose une construction innovante, il faut utiliser un modèle adapté (ici un modèle avec un poids de clef fixé),

— le filter permutator est particulièrement vulnérable aux attaques de type guess-and-determine.

Bien que la construction de filter permutator soit attirante pour le FHE, il est donc nécessaire de faire un effort de cryptanalyse de type guess-and-determine, et il apparaît que les modèles d'analyse usuels des chiffrements à flot ne sont pas toujours adaptés. Ici par exemple, un modèle à poids de clef fixé aurait été nécessaire, mais bien d'autres structures sont exploitables. Ce problème a été étudié à la suite de nos travaux par Carlet, Méaux et Rotella [CMR17], et je conseille sur ce point de se référer à la thèse (à paraître) de Yann Rotella.

Chapitre 5

Constructions pour les MACs

Sans contrefaçon, j'ai de la passion

Dans ce chapitre, j'évoque deux travaux très proches effectués avec Gaëtan Leurent [DL18a] en cours de publication. Il s'agit de travaux sur les MACs à bas coût.

Les MACs sont un autre aspect important de la cryptographie : là où les chiffrements servent à garantir la confidentialité, les MACs visent à garantir l'authenticité. Dans la pratique, il semble que les MACs sont généralement plus utiles pour les micro-contrôleurs que les chiffrements. Par exemple, un réseau de senseurs qui mesurent la température n'ont pas besoin de chiffrer leurs messages puisque la température est une donnée publique, mais les messages doivent être authentifiés pour éviter qu'une attaquante ne fausse les mesures. Un MAC peut aussi être utilisé pour une authentification entre une carte sans contact et un lecteur.

Or, bien que les MACs soient si importants pour les applications dans de petits terminaux comme les objets connectés, les solutions utilisées en pratique sont souvent peu sûres (par exemple MiFare, sur micro-contrôleurs, qui utilise une clef de seulement 48 bits, ou Keeloq, utilisé pour les voitures, dont la clef fait 60 bits), car les solutions existantes sont généralement trop coûteuses pour des objets avec des moyens aussi limités.

Nos travaux ont un intérêt double : d'une part, nous améliorons les preuves de sécurité des fonctions de hachage universelles (sur lesquelles se reposent beaucoup de constructions de MACs), et d'autre part nous construisons un nouveau MAC appelé XPMAC, que nous avons implémenté sur des micro-contrôleurs 32 bits avec d'excellentes performances.

5.1 Brève introduction aux MACs

Les Codes d'Authentification de Message (Message Authentication Codes, ou MACs) sont des primitives cryptographiques importantes, utilisées pour l'authentification des messages. Un MAC est une étiquette courte calculée par l'émetteur à partir du message et d'une clef secrète, envoyée avec le message et vérifiée par le receveur à l'aide de la même clef.

5.1.1 Construction de MACs

Les algorithmes de MAC peuvent être construits de diverses manières. L'une des premières propositions fut CBC-MAC [FIPS113], un algorithme basé sur un chiffrement par blocs, dont il existe des variantes plus récentes telles que OMAC [IK03]. Alternativement, PMAC [BR02] est un MAC basé sur un chiffrement par blocs parallélisable. Les algorithmes de MAC peuvent aussi être construits à partir de fonctions de hachage,

comme HMAC [BCK96], ou en partant de zéro, comme Siphash [AB12] ou Chaskey [Mou+14], qui est à l'heure actuelle le MAC avec l'implémentation la plus performante sur micro-contrôleurs 32 bits.

Toutes ces constructions sont déterministes, ce qui les rend simple d'utilisation, mais limite aussi leur sécurité. En effet, il existe une attaque générique pour forger contre tous les MACs déterministes itérés, qui utilise des collisions dans l'état interne, dû à Preneel et van Oorschot [Pv95]. De ce fait, ces MACs ne sont sûrs que jusqu'à la borne des anniversaires, *i.e.* lorsque la quantité de données authentifiée avec une même clef est bornée par $2^{n/2}$, avec n la taille de l'état.

Une façon d'améliorer la sécurité est d'utiliser un état interne plus grand. En particulier, les fonctions de hachage ont généralement un grand état interne ($n \geqslant 160$), et les MACs basés sur des fonctions de hachage offrent une sécurité suffisante en pratique même s'ils sont limités à $2^{n/2}$ blocs de message. Cette approche a aussi été utilisée récemment pour les MACs basés sur des chiffrements par blocs, avec des constructions telles que `SUM-ECBC` [Yas10], `3kf9` [Zha+12], et `PMAC+` [Dat+17] qui utilisent un état interne de $2n$ bits avec un chiffrement par blocs de n bits.

Une autre façon d'éviter l'attaque de Preneel et van Oorschot est de rendre le MAC non-déterministe, en utilisant soit un *nonce*, une valeur unique donnée par l'utilisateur (en pratique, le nonce est souvent un compteur), soit des jetons aléatoires choisies par l'algorithme de MAC, souvent appelés *sel*. Le nonce ou le sel est typiquement transmis avec le MAC pour vérification. Un exemple important de MAC basé sur un nonce est la construction de Wegman-Carter [WC81] qui authentifie un message M avec un nonce N par :

$$\mathrm{WC}[H, F]_{k_1,k_2}(M, N) = H_{k_1}(M) \oplus F_{k_2}(N),$$

avec H une famille de fonctions de hachage XOR universelles, et F une PRF [1]. Cette construction est sûre jusqu'à 2^n requêtes.

MACs à bas coût. Bien que les MACs semblent être une primitive importante pour la cryptographie à bas coût, peu de constructions ont été optimisées pour des environnements très contraints. Une exception notable est Chaskey [Mou+14], un MAC optimisé pour les micro-contrôleurs 32 bits avec d'excellentes performances logicielles.

5.1.2 Les fonctions de hachage [presque-]universelles

Les fonctions de hachage universelles ont été introduites par Carter et Wegman en 1977 [CW77], et ont été très largement utilisées depuis pour construire des MACs efficaces. En particulier, la construction Wegman-Carter-Shoup [WC81 ; Sho96] authentifie un message M avec un nonce N par :

$$\mathrm{WCS}[H, E]_{k_1,k_2}(M, N) = H_{k_1}(M) \oplus E_{k_2}(N),$$

avec H une famille de fonctions de hachage XOR universelles et E un chiffrement par blocs. Cette construction est utilisée dans GMAC, la fonction d'authentification de GCM [MV04], et dans Poly1305 [Ber05b], deux des schémas les plus largement utilisés dans TLS aujourd'hui [2].

Depuis l'introduction des fonctions de hachage universelles, un grand nombre de constructions ont été proposées pour construire des fonctions de hachage universelles efficaces, et pour transformer ces fonctions en MACs sécurisés.

1. PRF veut dire fonction pseudo-aléatoire, PRP veut dire permutation pseudo-aléatoire.
2. Les données de télémétrie de Mozilla montrent que plus de 90% des connexions HTTPS de Firefox 58 utilisent AES-GCM, et à peu près 0.5% utilisent ChaCha20-Poly1305 : `https://mzl.la/2GY53Mc` (accédé le 27 février 2018).

Les fonctions de hachage universelles ont un intérêt majeur : sur ces fonctions, on sait établir des preuves purement combinatoires sur les critères de sécurité. On a ainsi des preuves de sécurité robustes, contrairement au cas des constructions basées sur des chiffrements par blocs, qui reposent sur les arguments de sécurité obtenus par la cryptanalyse.

Définitions. Il existe plusieurs définitions liées aux fonctions de hachage universelles et presque universelles. En général, une fonction de hachage (presque) universelle H est une famille de fonctions (avec la notation $h \in H$, ou $H_k \in H$ pour insister sur la clef) telle que pour une fonction aléatoire de la famille, la probabilité (sur la clef) que deux entrées entraînent une collision en sortie soit toujours faible.

Dans le cas idéal, ceci donne la définition suivante :

Définition 5.1 (U)**.** Une famille de fonctions $H : A \to B$ est *universelle* (Universal) si :

$$\forall m_1 \neq m_2 \in A,\ |\{h \in H : h(m_1) = h(m_2)\}| \leqslant |H|/|B|$$

En pratique, on peut souvent tolérer une certaine déviation du cas idéal, avec un potentiel gain d'efficacité. C'est pourquoi on définit les fonctions de hachage presque universelles :

Définition 5.2 (ε-AU)**.** Une famille de fonctions $H : A \to B$ est *ε-presque universelle* (Almost Universal) si :

$$\forall m_1 \neq m_2 \in A,\ |\{h \in H : h(m_1) = h(m_2)\}| \leqslant \varepsilon|H|$$

Il est souvent utile de renforcer cette définition afin de couvrir une différence en sortie arbitraire, et pas uniquement le cas des collisions (une différence de 0). Ceci définit les fonctions presque XOR universelles :

Définition 5.3 (ε-AXU)**.** Une famille de fonctions $H : A \to B$ est *ε-presque* XOR *universelle* (Almost XOR-universal) si :

$$\forall m_1 \neq m_2 \in A,\ \forall d \in B,\ |\{h \in H : h(m_1) \oplus h(m_2) = d\}| \leqslant \varepsilon|H|$$

Si H est ε-AXU, elle est aussi ε-AU, et nous pouvons de surcroît définir une famille ε-AU $G : A \times B \to B$ de la façon suivante :

$$G = \{(m, b) \mapsto h(m) \oplus b : h \in H\}$$

Définition 5.4 (ε-ASU)**.** Une famille de fonctions $H : A \to B$ est *ε-presque fortement universelle* si :

$$\begin{cases} \forall m \in A,\ \forall b \in B,\ |\{h \in H : h(m) = b\}| = |H|/|B| \\ \forall m_1 \neq m_2 \in A, \forall b_1, b_2 \in B, |\{h \in H : h(m) = b_1, h(m_2) = b_2\}| \leqslant \varepsilon|H|/|B| \end{cases}$$

Si $H : A \to B$ est ε-ASU alors H est aussi ε-AU.

5.2 Constructions de MACs basées sur les fonctions de hachage universelles

Les fonctions de hachage universelles ont été introduites pour construire des MACs sécurisés, et sont dorénavant utilisés dans nombre de constructions de MAC et de preuves de sécurité. La première proposition par Wegman et Carter était de hacher un message et de chiffrer le résultat avec un masque jetable. Ceci définit un MAC avec une preuve de sécurité au sens de la théorie de l'information. En revanche, utiliser un masque jetable

n'est pas pratique, et il fut rapidement suggéré de le remplacer par la sortie d'une PRF, *i.e.* de le remplacer par le masque jetable par chiffrement en mode compteur.

La sécurité d'un algorithme de MAC est définie comme une borne supérieure sur la probabilité de succès d'un adversaire qui tente de forger une étiquette valide. Formellement, on considère un adversaire $\mathcal{A}$ avec un accès en oracle à l'algorithme de MAC F ; l'adversaire doit alors produire une paire message/étiquette, et réussit si le message n'a pas été soumis à l'oracle et que l'étiquette est valide. On définit l'avantage d'un tel adversaire avec un accès en oracle à l'algorithme de MAC par :

$$\mathbf{Adv}_F^{\mathrm{MAC}}(\mathcal{A}) = \Pr\left[F(M) = t : \mathcal{A}^F \to M, t\right] .$$

Similairement, on définit l'avantage pour distinguer d'une PRF ou d'une PRP d'un adversaire par :

$$\mathbf{Adv}_F^{\mathrm{PRP}}(\mathcal{A}) = \left|\Pr[\mathcal{A}^{\mathcal{F}} \to 1] - \Pr[\mathcal{A}^{\$} \to 1]\right| ,$$

où \$ représente une permutation aléatoire, et

$$\mathbf{Adv}_F^{\mathrm{PRF}}(\mathcal{A}) = \left|\Pr[\mathcal{A}^{\mathcal{F}} \to 1] - \Pr[\mathcal{A}^{\$} \to 1]\right| ,$$

où \$ représente une fonction aléatoire.

Nous notons aussi l'avantage maximal de tous les adversaires par $\mathbf{Adv}_F^{\mathrm{X}}$, où X est l'un de MAC, PRF, ou PRP. Un résultat important est le lemme d'échange entre PRF et PRP (PRF-PRP switching lemma), qui borne la sécurité d'un chiffrement par blocs lorsqu'il est utilisé comme une PRF [3] :

$$\mathbf{Adv}_E^{\mathrm{PRF}} \leqslant \mathbf{Adv}_E^{\mathrm{PRP}} + q^2/2^n$$

Dans cette section, nous passons en revue trois façons classiques de construire des MACs à partir de fonctions de hachage (presque) universelles ainsi que certaines propositions plus récentes. Par effort de simplicité, nous ferons l'hypothèse que l'adversaire fait q requêtes au MAC et une seule requête de vérification. Nous supposons que la fonction de hachage universel renvoie n bits, et que le MAC est aussi de taille n bits.

5.2.1 One-time MAC : $H(M)$

Ce nom de one-time MAC est une référence au « one-time-pad », le masque jetable. Si la clef est utilisée une seule fois, une famille ε-ASU peut être utilisée directement comme MAC [WC81], [Sti92, théorème 3.2]. En particulier, cette construction est utilisée dans le chiffrement authentifié ChaCha20-Poly1305 [NL15]. Si chaque clef n'est utilisée que pour authentifier un seul MAC, cette construction atteint une sécurité proche de 2^n :

$$\mathbf{Adv}_H^{\mathrm{MAC}} \leqslant \varepsilon + 2^{-n}$$

5.2.2 Construction Wegman-Carter : $H(M) \oplus F(N)$

La construction hacher puis masquer peut être utilisée pour authentifier plusieurs messages avec la même clef. En particulier, la construction Wegman-Carter utilise une famille de fonctions de hachage ε-AXU H et une PRF F :

$$\mathrm{WC}[H, F]_{k_1,k_2}(M, N) = H_{k_1}(M) \oplus F_{k_2}(N).$$

Tant que l'adversaire ne répète pas le nonce, cette construction atteint une sécurité de n bits :

$$\mathbf{Adv}_{\mathrm{WC}[H,F]}^{\mathrm{MAC}} \leqslant \mathbf{Adv}_F^{\mathrm{PRF}} + \varepsilon + 2^{-n}$$

3. Le chiffrement par blocs est une famille de permutations, pas de fonctions.

Dans la plupart des instanciations concrètes (GCM [MV04], Poly1305-AES [Ber05b]), la PRF est instanciée avec un chiffrement par blocs E, en suivant la construction de Wegman-Carter-Shoup [Sho96]. La sécurité peut être analysée en utilisant le PRF-PRP switching lemma, mais cela ajoute à la borne un terme $q^2/2^n$, correspondant à la borne du paradoxe des anniversaires. Cette preuve peut être améliorée en regardant directement la construction à base de PRP [Sho96 ; Ber05a], mais la sécurité est toujours limitée par la borne des anniversaires. En effet, il y a une attaque par distingueur simple qui consiste à requérir le MAC d'un même message fixé $2^{n/2}$ fois, et à vérifier s'il y a des collisions dans le MAC.

Afin d'éviter cette perte, on peut utiliser une construction pour transformer une PRP en PRF, telle que le XOR de deux permutations :

$$\text{WC-XoP}[H, E]_{k_1,k_2.k_3}(M, N) = H_{k_1}(M) \oplus E_{k_2}(N) \oplus E_{k_3}(N).$$

Comme le XOR de deux PRP est une PRF presque parfaite [Pat08 ; Pat13 ; DHT17], cette construction atteint n bits de sécurité.

5.2.3 Construction hacher puis PRF : $F(H(M))$

Alternativement, la construction hacher puis PRF construit un MAC déterministe à partir d'une famille ε-AU H et d'une PRF F :

$$\text{HF}[H, F]_{k_1,k_2}(M) = F_{k_2}(H_{k_1}(M)).$$

Cette construction a été analysée dans [Bla00, lemme 3.6.3] et [Bla+99] :

$$\mathbf{Adv}^{\text{MAC}}_{\text{HF}[H,F]} \leqslant \mathbf{Adv}^{\text{PRF}}_{F} + \frac{q^2}{2}\varepsilon + 2^{-n}$$

En particulier, plusieurs MAC basés sur des chiffrements par blocs, tels que CBC-MAC [BR00], peuvent être considérés comme des instances de la construction hacher puis PRF pour leur analyse. Plus récemment, cette technique a été utilisée pour les MACs avec sécurité au delà de la borne des anniversaires en utilisant une fonction de hachage sur $2n$ bits (SUM-ECBC [Yas10], PMAC+ [Yas11], 3kf9 [Zha+12], ZMAC [Iwa+17], ...).

La construction hacher puis PRF donne en fait une PRF (ce qui est plus fort qu'un MAC), et ceci est utilisé dans Periodic CBC Hash [MT06].

5.2.4 Hacher puis PRF avec nonce : $F(H(M)\|N)$

Certaines constructions permettent de combiner la sécurité de n bits avec des nonces, et une sécurité à la borne des anniversaires si le nonce est répété. En particulier, si une PRF sur $2n$ bits est disponible, on peut utiliser la construction introduite par UMAC [Bla+99] puis analysée sous le nom WMAC [BC09] avec une famille de fonctions ε-AU :

$$\text{WMAC}[H, F]_{k_1,k_2}(M, N) = F_{k_2}(H_{k_1}(M)\|N).$$

Cette construction a été analysée dans [Bla+99] sous l'hypothèse que les nonces sont toujours distincts :

$$\mathbf{Adv}^{\text{MAC}}_{\text{HFN}[H,F]} \leqslant \mathbf{Adv}^{\text{PRF}}_{F} + \varepsilon + 2^{-n}.$$

Si les nonces sont répétés, cette construction offre toujours une sécurité à la borne des anniversaires, en tant que cas particulier du schéma hacher puis PRF. Plus généralement, Black et Cochran ont donné une analyse en supposant qu'il y a une réutilisation des nonces limitée [BC09].

Nous notons qu'utiliser un chiffrement par blocs de $2n$ bits pour implémenter une PRF de $2n$ bits ne réduit pas significativement la sécurité, car la perte correspondante est seulement $q^2/2^{2n}$.

Récemment, des variantes de cette construction ont été proposées où la finalisation utilise un composant plus faible qu'une PRF sur $2n$ bits [CLS17] : la construction Nonce-as-Tweak utilise un chiffrement par blocs ajustable (tweakable block cipher), et la construction Nonce-as-Key utilise un chiffrement par blocs de n bits modélisé comme un chiffrement idéal.

5.2.5 EWCDM

Cogliati et Seurin ont récemment proposé une construction avec des propriétés de sécurité similaires utilisant seulement un chiffrement par blocs sur n bits [CS16] et une famille de fonctions ε-AXU :

$$\mathrm{EWCDM}[H,E]_{k_1,k_2,k_3}(M,N) = E_{k_3}\big(H_{k_1}(M) \oplus E_{k_2}(N) \oplus N\big).$$

Ils ont prouvé une sécurité jusqu'à $2^{2n/3}$ requêtes, et un travail ultérieur par Mennink et Neves [MN17] prouve une sécurité jusqu'à 2^n (en supposant que l'adversaire fait moins de $2^n/67$ requêtes) :

$$\mathbf{Adv}^{\mathrm{MAC}}_{\mathrm{EWCDM}[H,E]} \leqslant \mathbf{Adv}^{\mathrm{PRP}}_{F} + \frac{q}{2^n} + \frac{q^2\varepsilon}{2^n} + 2^{-n}$$

5.2.6 Discussion

Dans le contexte de la cryptographie à bas coût, les MACs basés sur des fonctions de hachage universelles sont séduisants car la fonction de hachage ne nécessite qu'une sécurité statistique et est généralement plus rapide à évaluer qu'une fonction cryptographique telle que CBC-MAC.

En pratique, la construction Wegman-Carter-Shoup est largement utilisée (en particulier dans GCM), mais ce n'est pas la meilleure manière de construire un MAC à partir d'une famille de fonctions de hachage universelles : cette construction requiert un nonce mais n'offre une sécurité que jusqu'à la borne des anniversaires. En particulier, une seule répétition de nonce casse souvent complètement le MAC [HP08]. À la place, nous nous focaliserons sur des constructions qui offrent une sécurité proche de 2^n requêtes quand le nonce est respecté, et qui peuvent tolérer une mauvaise utilisation des nonces. Suivant les primitives disponibles, nous recommandons d'utiliser soit la construction WMAC $F(H(M)\|N)$ avec un chiffrement par blocs sur $2n$ bits, soit la construction EWCDM avec un chiffrement par blocs sur n bits, soit la construction Nonce-as-Tweak avec un chiffrement par blocs adaptable sur n bits. L'inconvénient principal de ces constructions par rapport à la construction Wegman-Carter-Shoup est que le chiffrement par blocs ne peut pas être évalué en parallèle de la fonction de hachage, mais cela n'a pas vraiment de conséquences pour des implémentations sur micro-contrôleur.

Grâce à un mode qui offre une sécurité en 2^n, on peut utiliser des fonctions de hachage universelles avec une taille d'état de n bits plutôt que $2n$, ce qui permet de réduire le coût d'implémentation. Pour une application à bas coût, on pourra donc utiliser une famille de fonctions de hachage universelles dont la taille de sortie est $n = 64$.

5.3 Construction de fonctions de hachage universelles

Nous faisons ici un passage en revue des résultats précédents sur la construction de fonctions de hachage universelles.

5.3.1 Constructions pour des messages courts

De nombreuses constructions ont été proposées en vue de construire des fonctions de hachage universelles. En particulier, étant donné un corps $\mathbb{F}$, quelques exemples importants de fonctions de hachage universelles utilisent une multiplication par une clef secrète :

$$H_1 : \mathbb{F} \to \mathbb{F} = \{m \mapsto m \times k : k \in \mathbb{F}\} \qquad \text{est XOR universelle ;} \tag{5.1}$$

$$H_2 : \mathbb{F} \times \mathbb{F} \to \mathbb{F} = \{m_1, m_2 \mapsto m_1 \times k + m_2 : k \in \mathbb{F}\} \qquad \text{est universelle.} \tag{5.2}$$

Hachage polynomial [Die+92]. Afin de hacher des longs messages avec une seule clef, ces constructions peuvent être généralisés en un hachage polynomial. Le message d'entrée est interprété comme les coefficients d'un polynôme évalué en la clef secrète :

$$H : \mathbb{F}^\ell \to \mathbb{F} = \{(m_1, m_2, \dots m_\ell) \mapsto \sum_{i=1}^{\ell} m_i \times k^i : k \in \mathbb{F}\}.$$

La famille H avec des messages de longueur ℓ est $\ell\varepsilon$-AXU. Cette construction est utilisée pour construire des MACs de type Wegman-Carter pratiques comme GMAC et Poly1305, avec différents choix de corps $\mathbb{F}$.

Utiliser des chiffrements par blocs réduits [MT06]. En tant qu'alternative aux constructions par corps finis, les chiffrements par blocs réduits peuvent aussi être de bonnes fonctions de hachage universelles. En particulier, la valeur exacte du MEDP de 4 tours d'AES a été calculé par Keliher et Sui [KS07], et il est prouvé qu'il s'agit d'une famille ε-AXU avec $\varepsilon \approx 1.18 \cdot 2^{-110}$ (sous hypothèse que les sous-clefs de tour sont indépendantes) :

$$H : \{0,1\}^n \to \{0,1\}^n = \{x \mapsto E_k(x) : k \in \{0,1\}^n\} \qquad \text{est } \varepsilon\text{-AXU.} \tag{5.3}$$

Cette construction a été utilisée par Minematsu et Tsunoo pour construire un MAC basé sur AES plus rapide que CBC-MAC [MT06].

5.3.2 Composition et extension

Pour accepter des messages plus longs, plusieurs constructions de composition et d'extension peuvent être utilisées. Elle sont détaillées ci-dessous.

Produit cartésien [Sti92, Th 5.4]. Soit $H : A \to B$ une famille ε-AU. La construction par produit cartésien permet d'augmenter la taille de l'ensemble d'entrée et de l'ensemble de sortie en appliquant la fonction en parallèle. Plus précisément, la famille $G : A \times A \to B \times B$ est ε-AU :

$$G = \{(m_1, m_2) \mapsto (h(m_1), h(m_2)) : h \in H\}$$

Hachage par somme [CW77, Proposition 8] Soient $H_1 : A_1 \to B$ et $H_2 : A_2 \to B$ des familles ε-AXU. La construction par somme donne une famille G sur une entrée plus grande, mais avec une sortie de même taille. Plus précisément, $G : A_1 \times A_2 \to B$ est ε-AXU :

$$G = \{(m_1, m_2) \mapsto (h_1(m_1) \oplus h_2(m_2)) : h_1 \in H_1, h_2 \in H_2\}$$

Concaténation [Rog99, Proposition 3]. Soient $H_1 : A \to B_1$ une famille ε_1-AU et $H_2 : A \to B_2$ une famille ε_2-AU. La construction par concaténation donne une famille G avec une probabilité de collision réduite, au coût d'une taille de sortie accrue. Plus précisément, $G : A \to B_1 \times B_2$ est ε-AU, avec $\varepsilon = \varepsilon_1\varepsilon_2$:

$$G = \{m \mapsto (h_1(m), h_2(m)) : h_1 \in H_1, h_2 \in H_2\}$$

Composition [Sti92]. Soient $H_1 : A \to B$ et $H_2 : B \to C$ deux familles de fonctions de hachage presque universelles. On peut construire $G : A \to C$ comme la composition de H_1 et H_2 :

$$G = \{m \mapsto (h_2(h_1(m))) : h_1 \in H_1, h_2 \in H_2\}.$$

En particulier si H_1 et H_2 compressent leurs entrées, alors la composition compresse incrémentalement son entrée.

Plus précisément, on a les résultats suivants :

— Si H_1 est ε_1-AU et H_2 est ε_2-AU, alors G est ε-AU, avec $\varepsilon = \varepsilon_1 + \varepsilon_2 - \varepsilon_1\varepsilon_2 \leqslant \varepsilon_1 + \varepsilon_2$.

— Si H_1 est ε_1-AU et H_2 est ε_2-ASU, alors G est ε-ASU, avec $\varepsilon = \varepsilon_1 + \varepsilon_2 - \varepsilon_1\varepsilon_2 \leqslant \varepsilon_1 + \varepsilon_2$.

— Si H_1 est ε_1-AU et H_2 est ε_2-AXU, alors G est ε-AXU, avec $\varepsilon = \varepsilon_1 + \varepsilon_2 - \varepsilon_1\varepsilon_2 \leqslant \varepsilon_1 + \varepsilon_2$.

Les deux derniers résultats peuvent être utilisés pour composer une famille ε-AU efficace et une famille ε-AXU ou ε-ASU plus coûteuse.

Extension de domaine par composition. Les résultats précédents sur la composition peuvent être utilisés pour construire une famille de fonctions de hachage presque universelle de domaine d'entrée arbitraire à partir d'une famille de fonctions de hachage universelle compressive de taille fixe. Une construction naturelle est d'itérer la fonction de compression, comme la construction Merkle-Damgård pour les fonctions de hachage résistantes aux collisions.

Soient $H_1 : A_1 \to B_1$ et $H_2 : A_2 \times B_1 \to B_2$ ε_1-AU et ε_2-AU, respectivement. On définit l'itération de H_1 et H_2, $H : A_1 \times A_2 \to B_2$ de la façon suivante :

$$H = \{(m_1, m_2) \mapsto h_2(m_2, h_1(m_1)) : h_1 \in H_1, h_2 \in H_2\}$$

En utilisant les résultats précédents, on peut prouver que H est ε-AU avec $\varepsilon = \varepsilon_1 + \varepsilon_2 - \varepsilon_1\varepsilon_2 \leqslant \varepsilon_1 + \varepsilon_2$ car il s'agit de la composition de H_1' et H_2, où H_1' est aussi ε_1-AU :

$$H_1' : A_1 \times A_2 \to A_2 \times B_1 = \{(m_1, m_2) \mapsto (m_2, h_1(m_1)) : h_1 \in H_1\}.$$

De plus, si H_2 est ε_2-AXU (respectivement ε_2-ASU), alors H est aussi ε-AXU (respectivement ε-ASU).

Ce résultat peut aisément s'étendre à l'itération de trois fonctions ou plus. En particulier, on peut l'utiliser pour itérer une seule fonction ε-AU $H : B \times A \to B$, pour construire le ℓ-ème itéré $H^\ell : B \times A^\ell \to B$ avec ℓ clefs indépendantes ; H^ℓ est $\ell\varepsilon$-AU. En particulier, cette construction est utilisée dans [MT06].

Hachage par arbre [WC81]. Afin de réduire la taille de clef, Wegman et Carter décrivent une construction de hachage par arbre dans [WC81]. En partant d'une unique famille ε-AU $H : A \times A \to A$, une construction en arbre de profondeur d définit une famille $d\varepsilon$-AU $G_d : A^{2^d} \to A$ itérativement :

$$\begin{aligned} G_1 &= H \\ G_d &= \{(m_1, m_2) \mapsto h(g(m_1), g(m_2)) : h \in H, g \in G_{d-1}\} \end{aligned}$$

En utilisant les résultats précédents sur la composition et le produit cartésien, G_d est $d\varepsilon$-AU, et utilise seulement d clefs indépendantes.

Nous allons maintenant présenter deux nouveaux résultats Le premier porte sur l'extension de domaine des fonctions de hachage universelles lorsque certains composants sont bijectifs. Nous montrons que dans ce cas, on peut améliorer les preuves de sécurité. Le deuxième est la construction et l'implémentation efficace d'un nouveau MAC, XPMAC, conçu pour être efficace sur des micro-contrôleurs 32 bits, en conciliant les preuves robustes des fonctions de hachage universelles et les performances de Chaskey [Mou+14].

5.4 Extensions de domaine avec des permutations

La première partie de nos travaux porte sur les fonctions de hachage universelles. Cette section a pour but de montrer que les résultats d'extension de domaine de la littérature peuvent offrir une meilleure sécurité en ajoutant des conditions de bijectivité. En particulier, ceci améliore la preuve de sécurité du MAC de Minematsu et Tsunoo [MT06].

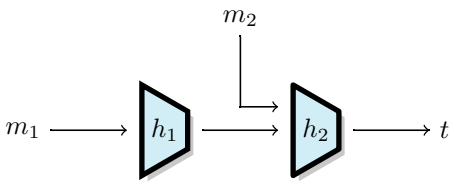


Figure 5.1 – Itération.

5.4.1 Itération avec une permutation

Notre premier résultat est un théorème de composition pour l'itération de deux familles AU (suivant la construction de la section 5.3.2) dans le cas particulier où la seconde fonction est une permutation lorsque la première entrée est fixée.

Nous montrons qu'avec cette condition additionnelle, l'itération de deux familles ε-AU est ε-AU, alors qu'elle n'est que ε'-AU avec $\varepsilon' = 2\varepsilon - \varepsilon^2$ en général. En effet, la probabilité de collision en sortie se décompose en : collision dans h_1 ou collision dans h_2, ces deux événements étant de probabilité ε. On a alors compté deux fois l'événement qu'il y ait collision en même temps dans h_1 et dans h_2, qui est de probabilité ε^2. Avec des permutations, on évite certains cas de collision, ce qui permet de réduire la probabilité de collision.

Théorème 5.1. *Soient $H_1 : A_1 \to B_1$ ε_1-AU et $H_2 : A_2 \times B_1 \to B_2$.*
Considérons $G : A_1 \times A_2 \to B_2$ définie telle qu'en figure 5.1, i.e.

$$G = \{(m_1, m_2) \mapsto h_2(m_2, h_1(m_1)) : h_1 \in H_1, h_2 \in H_2\}.$$

Si $x \mapsto h_2(m, x)$ est une permutation pour tous $h_2 \in H_2$ et $m \in A_2$, alors :
— *si H_2 est ε_2-AU, alors G est* $\max\{\varepsilon_1, \varepsilon_2\}$*-AU,*
— *si H_2 est ε_2-AXU, alors G est* $\max\{\varepsilon_1, \varepsilon_2\}$*-AXU,*
— *si H_2 est ε_2-ASU, alors G est* $\max\{\varepsilon_1, \varepsilon_2\}$*-ASU.*

En particulier, nous pouvons améliorer la borne de la preuve de sécurité de PC-MAC défini par Minematsu et Tsunoo [MT06], où un terme $d\varepsilon$ dû à l'itération de d fois 4 tours d'AES peut être remplacé par ε.

Notons que lorsque $x \mapsto h_2(m, x)$ n'est pas une permutation, il y a une perte d'entropie dans l'itération, et la sécurité de G est inférieure à la sécurité de H_1 et H_2.

Démonstration. **Cas 1 : $AU \Rightarrow AU$.**
Notons $N = \#\{(h_1, h_2) \in H_1 \times H_2 \mid h_2(m_2, h_1(m_1)) = h_2(m'_2, h_1(m'_1))\}$ pour une paire de messages fixée $(m_1 \| m_2, m'_1 \| m'_2)$. Nous voulons prouver que : $N \leqslant \max\{\varepsilon_1, \varepsilon_2\} \times |H_1| \times |H_2|$.

Si $m_2 = m'_2$ (et donc $m_1 \neq m'_1$), nous écrivons :

$$\begin{aligned} N &= \#\{(h_1, h_2) \in H_1 \times H_2 \mid h_2(m_2, h_1(m_1)) = h_2(m_2, h_1(m'_1))\} \\ &= \sum_{h_2 \in H_2} \#\{h_1 \in H_1 \mid h_2(m_2, h_1(m_1)) = h_2(m_2, h_1(m'_1))\} \\ &= \sum_{h_2 \in H_2} \#\{h_1 \in H_1 \mid h_1(m_1) = h_1(m'_1)\} \\ &\leqslant \sum_{h_2 \in H_2} \varepsilon_1 \times |H_1| = \varepsilon_1 \times |H_1| \times |H_2|. \end{aligned}$$

Nous avons utilisé que pour tout h_2, $x \mapsto h_2(m_2, x)$ est une permutation, donc $h_2(m_2, h_1(m_1)) = h_2(m'_2, h_1(m'_1))$ si et seulement si $h_1(m_1) = h_1(m'_1)$.
Si $m_2 \neq m'_2$, nous écrivons :

$$\begin{aligned} N &= \#\{(h_1, h_2) \in H_1 \times H_2 \mid h_2(m_2, h_1(m_1)) = h_2(m'_2, h_1(m'_1))\} \\ &= \sum_{h_1 \in H_1} \#\{h_2 \in H_2 \mid h_2(m_2, h_1(m_1)) = h_2(m'_2, h_1(m'_1))\} \\ &\leqslant \sum_{h_1 \in H_1} \varepsilon_2 \times |H_2| = \varepsilon_2 \times |H_1| \times |H_2|. \end{aligned}$$

Nous avons utilisé que $h_1(m_1)$ et $h_1(m'_1)$ sont des valeurs fixées pour chaque h_1 et $m_2 \neq m'_2$ fixés.
En fin de compte, nous obtenons que G est $\max\{\varepsilon_1, \varepsilon_2\}$-$AU$.

Cas 2 : $AXU \Rightarrow AXU$. Notons $N = \#\{(h_1, h_2) \in H_1 \times H_2 \mid h_2(m_2, h_1(m_1)) \oplus h_2(m'_2, h_1(m'_1)) = d\}$ pour une paire de messages fixée $(m_1 \| m_2, m'_1 \| m'_2)$ et un $d \in B_2$ fixé. Nous voulons prouver que $N \leqslant \max\{\varepsilon_1, \varepsilon_2\} \times |H_1| \times |H_2|$.
Le cas compliqué est la collision ($d = 0$), car la collision peut se produire soit dans h_1 soit dans h_2. En utilisant le cas AU, nous avons que lorsque $d = 0$, $N \leqslant \max\{\varepsilon_1, \varepsilon_2\} \times |H_1| \times |H_2|$.
Sinon, $d \neq 0$. Alors $(m_2, h_1(m_1)) \neq (m'_2, h_1(m'_1))$, et nous n'avons qu'à écrire :

$$\begin{aligned} N &= \#\{(h_1, h_2) \in H_1 \times H_2 \mid h_2(m_2, h_1(m_1)) \oplus h_2(m'_2, h_1(m'_1)) = d\} \\ &= \sum_{h_1 \in H_1} \#\{h_2 \in H_2 \mid h_2(m_2, h_1(m_1)) \oplus h_2(m'_2, h_1(m'_1)) = d\} \\ &\leqslant \sum_{h_1 \in H_1} \varepsilon_2 \times |H_2| = \varepsilon_2 \times |H_1| \times |H_2|. \end{aligned}$$

À nouveau, nous avons utilisé que, pour h_1 fixée, $h_1(m_1)$ et $h_1(m'_1)$ sont fixés.

Cas 3 : $ASU \Rightarrow ASU$. En premier lieu, il nous faut montrer que H est équilibrée. Pour des messages fixés m_1, m_2, et un $y \in B_2$ fixé, nous avons :

$$\begin{aligned} M &= \#\{(h_1, h_2) \in H_1 \times H_2 \mid h_2(m_2, h_1(m_1)) = y\} \\ &= \sum_{h_1 \in H_1} \#\{h_2 \in H_2 \mid h_2(m_2, h_1(m_1)) = y\} \\ &\leqslant \sum_{h_1 \in H_1} |H_2|/|B_2| = |H_1| \times |H_2|/|B_2|. \end{aligned}$$

Notons $N = \#\{(h_1, h_2) \in H_1 \times H_2 \mid h_2(m_2, h_1(m_1)) = y,\ h_2(m'_2, h_1(m'_1)) = y'\}$ pour une paire de messages fixée $(m_1 \| m_2, m'_1 \| m'_2)$ et $y, y' \in B_2$ fixés. Nous voulons prouver que $N \leqslant \max\{\varepsilon_1, \varepsilon_2\} \times |H_1| \times |H_2|$.

Similairement à la preuve du cas AXU, le cas compliqué est la collision ($y = y'$), car la collision peut se produire soit dans h_1 soit dans h_2. En utilisant le cas AU, nous obtenons que lorsque $y = y'$, $N \leqslant \max\{\varepsilon_1, \varepsilon_2\} \times |H_1| \times |H_2|$.
Sinon, $y \neq y'$. Alors $(m_2, h_1(m_1)) \neq (m'_2, h_1(m'_1))$, et nous écrivons simplement :

$$\begin{aligned} N &= \#\{(h_1, h_2) \in H_1 \times H_2 \mid h_2(m_2, h_1(m_1)) = y,\ h_2(m'_2, h_1(m'_1)) = y'\} \\ &= \sum_{h_1 \in H_1} \#\{h_2 \in H_2 \mid h_2(m_2, h_1(m_1)) = y,\ h_2(m'_2, h_1(m'_1)) = y'\} \\ &\leqslant \sum_{h_1 \in H_1} \varepsilon_2 \times |H_2| = \varepsilon_2 \times |H_1| \times |H_2|. \end{aligned}$$

À nouveau, nous avons utilisé que, pour h_1 fixée, $h_1(m_1)$ et $h_1(m'_1)$ sont fixés.

□

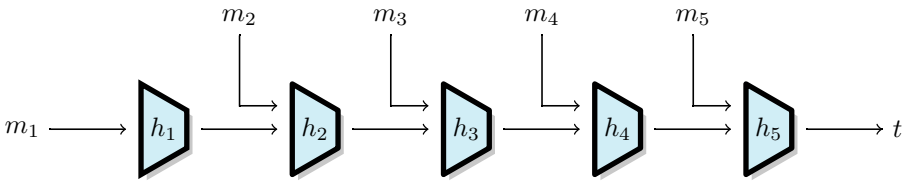


Figure 5.2 – Composition itérée.

Corollaire 5.1. *En appliquant le théorème 5.1 récursivement, on obtient que, pour $H_1, \ldots H_{\ell-1}$ des familles ε-AU. $G : A_1 \times \ldots \times A_\ell$ définie telle qu'en figure 5.2,* i.e.

$$G = \{m_1, m_2, \ldots m_\ell \mapsto h_\ell(m_\ell, h_{\ell-1}(m_{\ell-1}, \ldots)) : h_1 \in H_1, \ldots, h_\ell \in H_\ell\}$$

est $\max_{i \leqslant \ell}\{\varepsilon_i\}$*-AU (resp. AXU/ASU) si H_ℓ est ε_ℓ-AU (resp. AXU/ASU).*

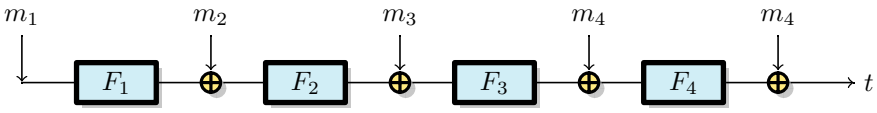


Figure 5.3 – Hachage chaîné.

Le théorème peut être utilisé pour étudier la sécurité de constructions basées sur une famille de permutations ε-AXU, comme les familles définies par la multiplication dans un corps (équation. (5.1)) ou un chiffrement par blocs réduit (équation (5.3)).

Corollaire 5.2 (Hachage chaîné utilisant des permutations)**.** *Soit H une famille de permutations ε-AXU sur A. On définit l'itération chaînée de H par (voir figure 5.3) :*

$$G = \{m_1, m_2, \ldots m_\ell \mapsto m_\ell \oplus h_\ell(m_{\ell-1} \oplus h_{\ell-2}(\ldots \oplus h_1(m_1))) : h_1 \in H, \ldots, h_\ell \in H\}.$$

Alors G est une famille ε-AU.

Le résultat vient du théorème 5.1 appliqué aux fonctions ε-AU suivantes :

$$H_i : A \times A \to A = \{(a, b) \mapsto b \oplus h(a) : h \in H\}$$

Grâce au théorème 5.1, on peut aussi obtenir une preuve alternative du hachage par somme de 2 ou ℓ fonctions de hachage universelles d'une famille ε-AXU.

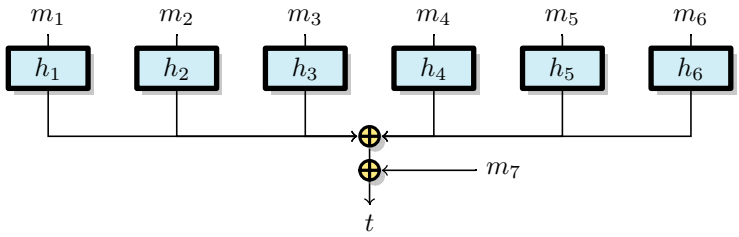


Figure 5.4 – Hachage par somme étendu.

Corollaire 5.3 (Hachage par somme étendu)**.** *Soit H une famille ε-AXU sur A (pas nécessairement de permutations). On définit l'itération par somme étendu de H par (voir figure 5.4) :*

$$G = \{m_1, m_2, \dots m_\ell \mapsto h_1(m_1) \oplus h_2(m_2) \oplus \dots \oplus h_{\ell-1}(m_{\ell-1}) \oplus m_\ell : h_1 \in H, \dots, h_\ell \in H\}.$$

Alors G est une famille ε-AU.

Les résultats viennent du théorème 5.1 appliqué aux fonctions suivantes :

$$H_i : A \times A \to A = \{(a, b) \mapsto a \oplus h(b) : h \in H\}$$

qui sont ε-AU. Nous donnons aussi des résultats plus détaillés sur le hachage par somme étendu, en particulier qu'en retirant un dernier $\oplus$, la famille obtenue est ε-AXU (résultat déjà connu, cf. section 5.3.1) et des preuves de sécurité plus détaillées.

Théorème 5.2 (Sécurité du hachage par somme étendu)**.** *Soient $H_1 : A_1 \to B$, $H_2 : A_2 \to B$.*

Considérons $H : A_1 \times A_2 \to B$ définie par :

$$H = \{(h_1, h_2) \in H_1 \times H_2 \mid h(m_1, m_2) = h_1(m_1) \oplus h_2(m_2), m_1 \in A_1, m_2 \in A_2\}.$$

Alors nous avons les résultats de sécurité suivants :
- *si H_1 est ε_1-AU et H_2 est ε_2-AXU, alors h est $\max\{\varepsilon_1, \varepsilon_2\}$-AU,*
- *si H_1 est ε_1-AXU et H_2 est ε_2-AXU, alors h est $\max\{\varepsilon_1, \varepsilon_2\}$-AXU,*
- *si H_1 est ε_1-ASU et H_2 est ε_2-ASU, alors h est $\max\{\varepsilon_1, \varepsilon_2\}$-ASU.*

Démonstration. Fixons $m_1, m_1' \in A_1$, $m_2, m_2' \in A_2$.

Cas 1 : *AU* Nous avons H_1 ε_1-AU, H_2 ε_2-AXU.

Ce résultat peut être directement déduit du cas $AU \Rightarrow AU$ du théorème 5.1, en considérant la famille $H_2' = \{(m_1, m_2) \mapsto m_1 \oplus h_2(m_2) : h_2 \in H_2, m_1 \in A_1, m_2 \in A_2\}$, car H est alors $\{(h_1, h_2') \in H_1 \times H_2' \mid h(m_1, m_2) = h_2(h_1(m_1, m_2), m_1 \in A_1, m_2 \in A_2\}$. Afin de voir le lien direct avec les cas 2 et 3, nous donnons ici une preuve alternative.

Si $m_2 = m_2'$, nous utilisons que H_1 est ε_1-AU :

$$\begin{aligned}
&\#\{(h_1, h_2) \in H_1 \times H_2 \mid h_1(m_1) \oplus h_2(m_2) = h_1(m_1') \oplus h_2(m_2')\} \\
&= \sum_{h_2 \in H_2} \#\{h_1 \in H_1 \mid h_1(m_1) = h_1(m_1')\} \\
&\leqslant \sum_{h_2 \in H_2} \varepsilon_1 \times |H_1| \\
&= \varepsilon_1 \times |H_1| \times |H_2|.
\end{aligned}$$

Si $m_2 \neq m_2'$, nous utilisons que H_2 est ε_2-AXU :

$$\begin{aligned}
&\#\{(h_1,h_2) \in H_1 \times H_2 \mid h_1(m_1) \oplus h_2(m_2) = h_1(m_1') \oplus h_2(m_2')\} \\
&= \sum_{h_1 \in H_1} \#\{h_2 \in H_2 \mid h_2(m_2) \oplus h_2(m_2') = h_1(m_1) \oplus h_1(m_1')\} \\
&\leqslant \sum_{h_1 \in H_1} \varepsilon_2 \times |H_2| \\
&= \varepsilon_2 \times |H_1| \times |H_2|.
\end{aligned}$$

Cas 2 : *AXU* Fixons $d \in B$.

Nous avons H_1 ε_1-AXU, H_2 ε_2-AXU.

Si $m_2 = m_2'$ et $m_1 = m_1'$, nous utilisons le cas AU (cas 1).

Si $m_2 = m_2'$ (et donc $m_1 \neq m_1'$), nous utilisons que H_1 est ε_1-AXU :

$$\begin{aligned}
&\#\{(h_1,h_2) \in H_1 \times H_2 \mid h_1(m_1) \oplus h_2(m_2) \oplus h_1(m_1') \oplus h_2(m_2') = d\} \\
&= \sum_{h_2 \in H_2} \#\{h_1 \in H_1 \mid h_1(m_1) \oplus h_1(m_1') = d \oplus h_2(m_2) \oplus h_2(m_2')\} \\
&\leqslant \sum_{h_2 \in H_2} \varepsilon_1 \times |H_1| \\
&= \varepsilon_1 \times |H_1| \times |H_2|.
\end{aligned}$$

Si $m_2 \neq m_2'$, symétriquement, nous utilisons que H_2 est ε_2-AXU et obtenons que

$$\#\{(h_1,h_2) \in H_1 \times H_2 \mid h_1(m_1) \oplus h_2(m_2) \oplus h_1(m_1') \oplus h_2(m_2') = d\} \leqslant \varepsilon_2 \times |H_1| \times |H_2|.$$

Cas 3 : *ASU* Fixons $u, u' \in B$.

Nous avons H_1 ε_1-ASU, H_2 ε_2-ASU.

Si $m_2 = m_2'$, nous utilisons que H_1 est ε_1-ASU :

$$\begin{aligned}
&\#\{(h_1,h_2) \in H_1 \times H_2 \mid h_1(m_1) \oplus h_2(m_2) = u,\, h_1(m_1') \oplus h_2(m_2') = u'\} \\
&= \sum_{h_2 \in H_2} \#\{h_1 \in H_1 \mid h_1(m_1) = u \oplus h_2(m_2),\, h_1(m_1') = u' \oplus h_2(m_2')\} \\
&\leqslant \sum_{h_2 \in H_2} \varepsilon_1 \times |H_1| \\
&= \varepsilon_1 \times |H_1| \times |H_2|.
\end{aligned}$$

Si $m_2 \neq m_2'$, symétriquement, nous utilisons que H_2 est ε_2-ASU et obtenons que

$$\begin{gathered}
\#\{(h_1,h_2) \in H_1 \times H_2 \mid h_1(m_1) \oplus h_2(m_2) = u,\, h_1(m_1') \oplus h_2(m_2') = u'\} \\
\leqslant \\
\varepsilon_2 \times |H_1| \times |H_2|\,.
\end{gathered}$$

Il nous faut aussi prouver que, $\forall (m_1,m_2) \in A_1 \times A_2$, $\forall b \in B$,

$$\#\{(h_1,h_2) \in H_1 \times H_2 : h_1(m_1) \oplus h_2(m_2) = b\} = |H_1||H_2|/|B|\,.$$

On a, $\forall (m_1,m_2) \in A_1 \times A_2$, $\forall b \in B$,

$$\begin{aligned}
&\#\{(h_1,h_2) \in H_1 \times H_2 : h_1(m_1) \oplus h_2(m_2) = b\} \\
&= \sum_{h_2 \in H_2} \{h_1 \in H_1 : h_1(m_1) = b \oplus h_2(m_2)\} \\
&= \sum_{h_2 \in H_2} \tfrac{|H_1|}{|B|} \\
&= \tfrac{|H_1||H_2|}{|B|}\,.
\end{aligned}$$

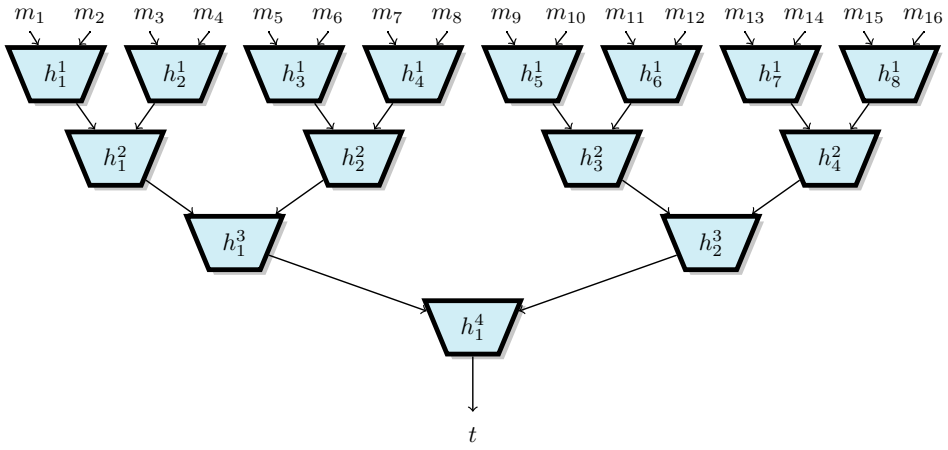


Figure 5.5 – Structure générale du hachage par arbre. S_i représente S utilisant k_i pour clef.

Nous avons utilisé le fait que $\#\{h_1 \in H_1 : h_1(m_1) = b'\} = |H_1|/|B|$, puisque H_1 est ε-ASU.

□

Ceci se généralise directement à plus de deux branches comme au corollaire 5.3.

5.4.2 Modification du hachage par arbre pour atteindre une meilleure sécurité

Considérons désormais la construction de hachage par arbre de [WC81] (cf. figure 5.5). Dans [WC81], les auteurs introduisent ce schéma afin de hacher un message avec une clef de longueur logarithmique en la longueur du message, en utilisant la même clef pour toutes les fonctions au même étage dans l'arbre. Nous montrons qu'en utilisant des permutations et des clefs indépendantes deux à deux à chaque étage, nous atteignons une sécurité meilleure que la version originale en gardant la longueur de clef logarithmique.

Considérons une famille ε-AU $H : \mathbb{F} \times \mathbb{F} \to \mathbb{F}$ telle que $x \mapsto h(x, y)$ et $y \mapsto h(x, y)$ soient des permutations pour tout $h \in H$, avec $\mathbb{F}$ un corps fini. Nous définissons une construction en arbre où chaque étage utilise une variante de la construction par produit cartésien avec des fonctions de hachage de H deux à deux indépendantes, et où les fonctions de hachage sont indépendantes entre les étages. Plus précisément, au niveau i nous utilisons la clef $k_i \in \mathbb{F}^2$ et nous appliquons une fonction de hachage $H^d_{k_i}$ définie comme l'application parallèle des fonctions h^i_j avec $h^i_j = H_{k_i[0]+j\times k_i[1]}$. Formellement, nous fixons la profondeur totale d, et nous définissons H^i pour le i-ème étage, et G^i comme la composition des i premiers étages :

$$\begin{aligned} H^i_k : \mathbb{F}^{2^{d-i+1}} \to \mathbb{F}^{2^{d-i}} \qquad & (x_1, x_2, x_3, \ldots) \mapsto (h^i_1(x_1, x_2), h^i_2(x_3, x_4), \ldots) \\ & \text{avec } h^i_j = H_{k_i[0]+j\times k_i[1]} \\ G^i_{k_1,\ldots,k_i} : \mathbb{F}^{2^d} \to \mathbb{F}^{2^{d-i}} \qquad & (x_1, x_2, x_3, \ldots) \mapsto H^i_{k_i}(\ldots H^2_{k_2}(H^1_{k_1}(x_1, x_2, \ldots))) \\ & G^i_{k_1,\ldots,k_i} = H^i_{k_i} \circ G^{i-1}_{k_1,\ldots,k_{i-1}} \end{aligned}$$

Enfin, nous définissons la famille $G^d = \{G^d_k : k \in \mathbb{F}^{2d}\}$.

Théorème 5.3 (Sécurité du hachage par arbre amélioré)**.** *Avec la construction précédente, si H est une famille ε-AU, alors G^d est une famille ε'-AU, avec $\varepsilon' = \varepsilon + (d-1)^2 \cdot \varepsilon^2$.*

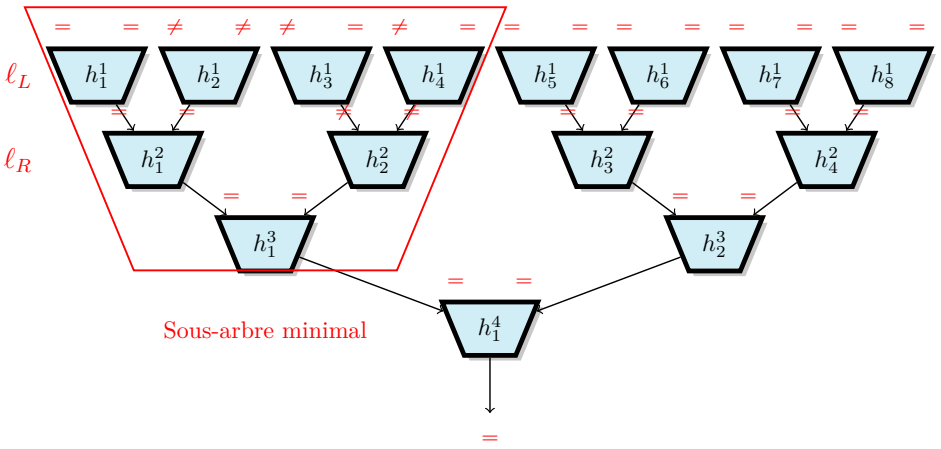


Figure 5.6 – Exemple de propagation des différences et sous-arbre minimal avec des différences pour le hachage par arbre.
$=$ représente une différence de nulle, $\neq$ représente une différence non-nulle.

Notons que la construction générale de hachage par arbre ne donnerait qu'une famille $d\varepsilon$-AU.

Démonstration. Fixons deux messages d'entrée $M = (m_1, \ldots, m_a)$ et $M' = (m'_1, \ldots, m'_a)$, avec $M \neq M'$. Remarquons que, puisque $x \mapsto h_j^i(x, y)$ et $y \mapsto h_j^i(x, y)$ sont des permutations de $\mathbb{F}_2^n$ dans $\mathbb{F}_2^n$, une fonction avec une différence de zéro en sortie doit avoir soit une différence de zéro en entrée, soit une différence dans ses deux entrées. Sans perte de généralité, supposons que chacun des sous-arbres gauche et droit de G^d ont des différences dans leurs entrées. Autrement, nous restreignons l'analyse au sous-arbre minimal de G^d avec des différences en entrée, et nous savons que la sortie du sous-arbre doit collisionner et que les deux moitiés ont des différences en entrée (cf. figure 5.6).

Nous voulons borner le nombre de clefs N telles que $G_k^d(M) = G_k^d(M')$. Pour commencer, définissons les moitiés droite et gauche de l'état de chaque étage dans l'arbre par $G_k^{i,R}(M)$ et $G_k^{i,L}(M)$, avec $G_k^i(M) = G_k^{i,L}(M) \| G_k^{i,R}(M)$. Pour une clef k donnée, nous définissons aussi ℓ_R et ℓ_L en tant que derniers étages du sous-arbre avec une différence en sortie :

$$\ell_L(k) = \max\{i < d : G_k^{i,L}(M) \neq G_k^{i,L}(M')\}$$
$$\ell_R(k) = \max\{i < d : G_k^{i,R}(M) \neq G_k^{i,R}(M')\}$$

Ceci nous permet de diviser les clefs par rapport aux valeurs de $\ell(k)$, et de borner le nombre de clefs dans chaque sous-ensemble :

$$\begin{aligned} N &= \#\{k : G_k^d(M) = G_k^d(M')\} \\ &= \sum_{i,j \in \{0,\ldots,d-1\}} \#\{k : G_k^d(M) = G_k^d(M'), \ell_L(k) = i, \ell_R(k) = j\} \end{aligned}$$

Remarquons d'abord que si $\ell_L(k) = d - 1$ et $G_k^d(M) = G_k^d(M')$, alors il faut aussi que $\ell_R(k) = d - 1$ car $x \mapsto h_1^1(x, y)$ est une permutation (respectivement, si $\ell_R(k) = d - 1$ alors $\ell_L(k) = d - 1$).

Cas 1 : $\ell_L(k) = \ell_R(k) = d-1$. Nous partons du cas où les différences s'annulent dans le dernier nœud de l'arbre.

$$\begin{aligned} N_1 &= \#\{k : G_k^d(M) = G_k^d(M'), \ell_L(k) = \ell_L(k) = d-1\} \\ &= \#\{k_1, k_2, \ldots, k_{d-1} : G_k^{d-1}(M) \neq G_k^{d-1}(M')\} \\ &\quad \times \#\{k_d : H_k^d(G_k^{d-1}(M)) = H_{k_d}^d(G_k^{d-1}(M'))\} \\ &\leqslant |\mathbb{F}|^{2(d-1)} \times \varepsilon \cdot |\mathbb{F}|^2 \end{aligned}$$

La dernière inégalité vient du fait que pour tout choix valide de $k_1, k_2, \ldots, k_{d-1}$, $G_k^{d-1}(M)$ et $G_k^{d-1}(M')$ sont des valeurs fixées avec $G_k^{d-1}(M) \neq G_k^{d-1}(M')$, et $H_{k_d}^d$ dépend seulement de k_d. Ainsi, seule une fraction ε des choix de k_d satisfont la condition.

Cas 2 : $\ell_L(k) = \ell_R(k) < d-1$. Ensuite, considérons les clefs avec $\ell_L(k) = \ell_R(k) = i$ avec $0 \leqslant i < d-1$.

$$\begin{aligned} N_2 &= \#\{k : G_k^d(M) = G_k^d(M'), \ell_L(k) = \ell_R(k) = i\} \\ &= \#\{k_1, k_2, \ldots, k_i : G_k^{i,L}(M) \neq G_k^{i,L}(M'), G_k^{i,R}(M) \neq G_k^{i,R}(M')\} \\ &\quad \times \#\{k_{i+1} : H_{k_{i+1}}^{i+1}(G_k^i(M)) = H_{k_{i+1}}^{i+1}(G_k^i(M'))\} \times \#\{k_{i+2}, \ldots k_d\} \\ &\leqslant |\mathbb{F}|^{2(d-1)} \times \varepsilon^2 \cdot |\mathbb{F}|^2 \end{aligned}$$

À nouveau, $G_k^i(M)$ et $G_k^i(M')$ sont des valeurs fixées pour tout choix de $k_1, k_2, \ldots, k_i$, et $H_{k_{i+1}}^{i+1}$ dépend uniquement de k_{i+1}. De plus, puisque nous considérons les clefs $k_1, k_2, \ldots, k_i$ avec $G_k^{i,L}(M) \neq G_k^{i,L}(M')$ et $G_k^{i,R}(M) \neq G_k^{i,R}(M')$, il y a deux fonctions h_u^{i+1} et h_v^{i+1} avec une différence en entrée non-nulle et une différence en sortie nulle. Ceci restreint les clefs possibles pour k_{i+1} à une fraction $\ll \varepsilon^2$ car les clefs correspondantes sont indépendantes.

Cas 3 : $\ell_L(k) \neq \ell_R(k)$. Pour finir, nous considérons les clefs avec $\ell_L(k) = i, \ell_R(k) = j$ avec $0 \leqslant i, j < d-1$, $i \neq j$. Sans perte de généralité, nous supposons $i < j$.

$$\begin{aligned} N_3 &= \#\{k : \ell_L(k) = i, \ell_R(k) = j\} \\ &= \#\{k_1, k_2, \ldots, k_i : G_k^{i,L}(M) \neq G_k^{i,L}(M'), G_k^{i,R}(M) \neq G_k^{i,R}(M')\} \\ &\quad \times \#\{k_{i+1} : H_{k_{i+1}}^{i+1,L}(G_k^{i,L}(M)) = H_{k_{i+1}}^{i+1,L}(G_k^{i,L}(M')), G_k^{i+1,R}(M) \neq G_k^{i+1,R}(M')\} \\ &\quad \times \#\{k_{i+2}, k_{i+3}, \ldots, k_j : G_k^{j,R}(M) \neq G_k^{j,R}(M')\} \\ &\quad \times \#\{k_{j+1} : H_{k_{j+1}}^{j+1,R}(G_k^{j,R}(M)) = H_{k_{j+1}}^{j+1,R}(G_k^{j,R}(M'))\} \times \#\{k_{j+2}, \ldots k_d\} \\ &\leqslant |\mathbb{F}|^{2i} \times \varepsilon \cdot |\mathbb{F}|^2 \times |\mathbb{F}|^{2(j-i-1)} \times \varepsilon \cdot |\mathbb{F}|^2 \times |\mathbb{F}|^{2(d-j-1)} \end{aligned}$$

En effet, après avoir choisi $k_1, \ldots k_i$, seule une fraction au plus ε des choix pour k_{i+1} sont valides, et après avoir choisi en plus $k_{i+2}, \ldots k_j$, seule une fraction au plus ε des choix pour k_{j+1} sont valides.

Finalement nous pouvons conclure :

$$\begin{aligned} N &= N_1 + (d-1)N_2 + (d-1)(d-2)N_3 \\ &\leqslant (\varepsilon + (d-1)^2 \cdot \varepsilon^2) \cdot |\mathbb{F}|^{2d}. \end{aligned}$$

□

5.4.3 Conclusion

Ces deux résultats prouvent que certaines constructions pour étendre le domaine des familles fonctions de hachage universelles peuvent être améliorées sous l'hypothèse que les fonctions de hachage universelles utilisées comme composants élémentaires sont des permutations.

5.5 Construction d'un MAC à bas coût : XPMAC

L'objectif de ce travail est d'obtenir un MAC qui ait les bonnes propriétés de sécurité des MACs basés sur les fonctions de hachage universelles et les bonnes propriétés de performances de Chaskey [Mou+14] sur micro-contrôleurs 32 bits (qui est à l'heure actuelle le MAC le plus performant sur micro-contrôleurs 32 bits).

Nous proposons donc une instanciation de fonction de hachage universelle pour micro-contrôleurs et un MAC correspondant. Tel qu'expliqué dans la section 5.2.6, nous nous orientons vers une construction à base de nonce avec des propriétés acceptables en cas de mauvaise utilisation des nonces (telles que WMAC, EWCDM ou Nonce-as-Tweak), en utilisant une fonction de hachage universelle avec une sortie sur 64 bits.

Une partie importante de ce travail consiste en l'implémentation et l'optimisation de notre algorithme, XPMAC, sur deux micro-contrôleurs 32 bits. Nous parvenons à obtenir des performances similaires à celles de Chaskey en utilisant des fonctions de hachage universelles (un hachage polynomial) et le mode WMAC.

En particulier, nous avons implémenté une optimisation poussée de la multiplication sur $\mathbb{F}_{2^{128}}$ et sur $\mathbb{F}_{2^{64}}$. La multiplication sur $\mathbb{F}_{2^{128}}$ est le composant principal de GMAC, aussi notre code peut être utilisé pour optimiser GMAC sur un micro-contrôleur 32 bits. Une version de GMAC optimisée obtient alors des performances seulement deux fois moins bonnes. L'avantage de XPMAC est de travailler sur un état de 64 bits et non 128 bits (grâce à l'utilisation d'un mode au delà de la borne des anniversaires, WMAC), ce qui permet d'obtenir ces meilleures performances.

Environnements de test. Afin d'évaluer divers choix possibles de construction et de comparer notre schéma avec d'autres algorithmes de MAC, nous avons effectué des tests d'implémentations sur deux micro-contrôleurs 32 bits : une carte FRDM-KL46Z avec un micro-contrôleur Cortex-M0+ (à une fréquence de 48MHz, avec 32KB de SRAM) et une carte FRDM-K64F avec un micro-contrôleur Cortex-M4 (à une fréquence de 120MHz, avec 256KB de RAM). Le Cortex-M0+ est un micro-contrôleur très limité, alors que le Cortex-M4 est légèrement plus puissant, avec une plus grande quantité de RAM et un jeu d'instructions plus fourni. Pour les évaluations, nous avons utilisé le compteur de cycles du Cortex-M4, mais comme notre Cortex-M0+ n'en contient pas, nous avons utilisé un compteur (en secondes) et nous l'avons divisé par la fréquence.

5.5.1 Choix de la fonction de hachage universelle : XPoly

Nous nous concentrons sur les fonctions de hachage universelles basées sur des propriétés combinatoires : elles offrent généralement une excellente performance et une sécurité au niveau de la théorie de l'information. En particulier, il existe de nombreuses constructions basées sur l'arithmétique dans les corps finis qui offrent un vaste compromis entre la taille de clef et la sécurité du hachage, et diverses options d'implémentation. L'objectif reste de créer un MAC performant sur micro-contrôleur 32 bits, si possible aussi performant que Chaskey.

Compromis entre la taille de clef et la sécurité. Il y a habituellement un compromis entre la taille de clef et la sécurité d'une fonction de hachage universelle. En particulier, il y a deux cas extrêmes :

le hachage polynomial [Die+92] : $H_k : m_1, \dots m_\ell \mapsto \sum_{i=1}^{\ell} m_i \times k^i$
H_k est une famille $\ell\varepsilon$-AXU utilisant un seul élément du corps pour clef.

le produit scalaire [GMS74] : $H_{k_1,\dots k_\ell} : m_1, \dots m_\ell \mapsto \sum_{i=1}^{\ell} m_i \times k_i$
$H_{k_1,\dots k_\ell}$ est une famille ε-AXU utilisant ℓ éléments du corps fini pour clef.

En particulier, le facteur ℓ du hachage polynomial donne des classes de clefs faibles pour GCM [PC14].

XOR de polynômes. Afin d'équilibrer sécurité et taille de clef, nous utilisons une construction basée sur une somme de polynômes. Plus précisément, nous divisons le message en morceaux de longueur λ, et hachons chaque morceau séparément avec des hachages polynomiaux utilisant des clefs indépendantes. Ensuite, nous sommons leurs résultats (voir figure 5.7). Nous appelons cette construction XPoly. Comme les hachages polynomiaux sont $\lambda\varepsilon$-AXU, XPoly est aussi une famille $\lambda\varepsilon$-AXU, d'après l'analyse du hachage par somme étendu (cf. théorème 5.2).

$$\mathrm{XPoly}_k : m_1, \dots m_\ell \mapsto \sum_{i=1}^{\lceil \ell/\lambda \rceil} \sum_{j=1}^{\lambda} m_{\lambda i+j} \times k_i^j$$

Le paramètre λ offre un compromis entre la sécurité et la longueur de clef. La longueur de la clef est linéaire en la taille du message, mais nous pouvons utiliser une PRF pour étendre la clef-maître en sous-clefs utilisées pour chaque morceau, avec $k_i = F_k(i)$. Si λ n'est pas trop petit, le temps que prend la dérivation de clef reste faible.

Cette construction évite la perte de sécurité du hachage polynomial pour les longs messages, sans la complexité additionnelle d'une construction par arbre. À vrai dire, les taille de clef et sécurité obtenues sont comparables pour les petites tailles de messages généralement utilisées dans les environnements contraints. Par exemple, pour traiter des messages avec $\ell = 256$ blocs (2kB avec $n = 64$, ce qui est plus grand que la taille usuelle d'un paquet IP), une construction par arbre classique requiert 8 clefs et atteint une sécurité de 8ε ; avec 8 clefs, notre construction atteint une sécurité de 32ε, mais elle est plus simple à implémenter et plus flexible.

5.5.2 Choix du corps et multiplication.

Il nous faut désormais choisir le corps pour définir notre fonction de hachage universelle. Il y a deux types de corps qui peuvent être utilisés pour des implémentations efficaces : les corps $\mathbb{F}_p$ définis modulo un nombre premier p proche de 2^{64} ou 2^{128} (tel qu'utilisé dans Poly1305), et les corps binaires tels que $\mathbb{F}_{2^{64}}$ et $\mathbb{F}_{2^{128}}$ (comme dans GCM).

Le choix du corps est essentiel : en effet, l'opération coûteuse de XPoly est la multiplication dans le corps. Choisir un corps adapté permet d'avoir une multiplication peu coûteuse. Nous choisissons un corps binaire ($\mathbb{F}_{2^{64}}$ en l'occurrence, puisque nous visons un MAC avec un état de 64 bits, mais on peut envisager de choisir $\mathbb{F}_{2^{128}}$), et nous montrons qu'il est possible d'implémenter la multiplication dans un tel corps à un faible coût sur micro-contrôleur 32 bits.

Nous utilisons une implémentation par tables de multiplication. Cette technique a déjà été employée pour des implémentations de GCM sans contraintes de bas coût, et une version simple (avec des morceaux de 1 bits) est utilisée dans l'implémentation de GCM de `mbed TLS`, une librairie TLS pour micro-contrôleurs.

Implémentation par tables. Comme la multiplication par une clef fixée est une opération linéaire, elle peut être implémentée en précalculant des tables. Par exemple, si l'on précalcule $k_i = 2^i \times k$ pour $0 \leqslant i < n$, on peut décomposer un élément $x \in \mathbb{F}$ en $x = \sum_{1 \leqslant i \leqslant n} x_i \times 2^i$ (où x_i est simplement le i-ème bit de x), et on utilise :

$$x \times k = \sum_{1 \leqslant i \leqslant n} x_i \times 2^i \times k = \sum_{1 \leqslant i \leqslant n} x_i \times k_i.$$

En particulier, sur un corps binaire, la somme est un simple XOR qui peut être évalué facilement. Cette technique est généralement appelée algorithme de Shoup [Sho99].

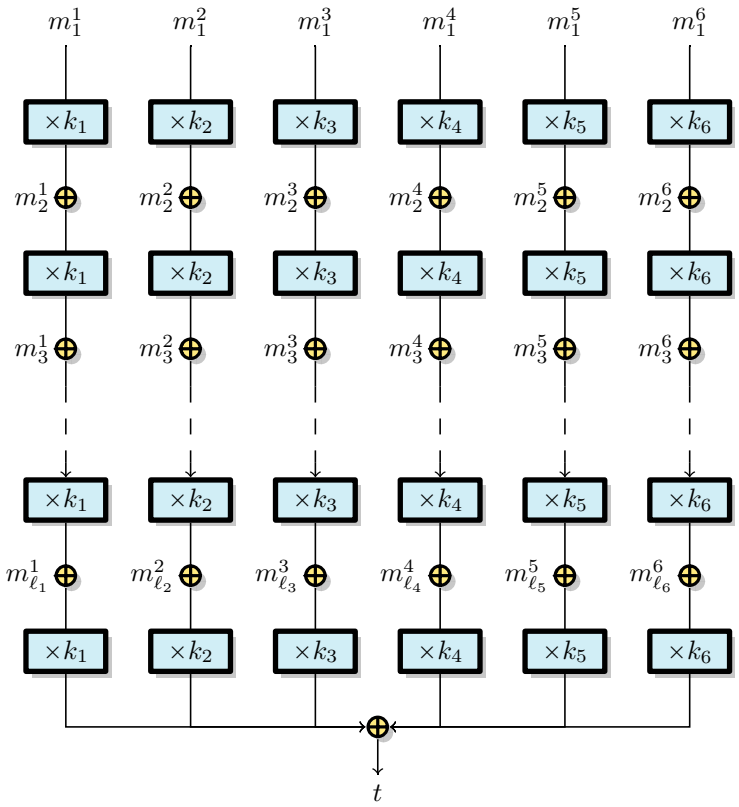


Figure 5.7 – XPoly : une fonction de hachage universelle basée sur le hachage par XOR et des multiplications dans le corps fini.

Plus généralement, on peut calculer les tables de multiplications pour plusieurs bits consécutifs. Si l'on divise x en morceaux de t bits et qu'on précalcule des tables à 2^t entrées pour chaque morceau, il nous faut uniquement n/t accès a des tables et $n/t - 1$ sommes pour évaluer le produit $x \times k$.

Implémentation par multiplication. Alternativement, sur un processeur puissant, la multiplication dans un corps fini défini modulo un premier peut être implémentée efficacement en utilisant le multiplicateur d'entiers du processeur. Ceci est important pour les serveurs utilisant des clefs différentes avec plusieurs clients, car accéder à des tables provoque des *cache misses*, comme argumenté dans [Ber05b].

Évaluations. Nous donnons les résultats d'évaluation de la multiplication dans plusieurs corps, avec plusieurs stratégies d'implémentation dans la table 5.1. Pour référence, nous utilisons une implémentation existante de Poly1305 (multiplication dans $\mathbb{F}_{2^{130}-5}$) et de GMAC (multiplication dans $\mathbb{F}_{2^{128}}$). Nous utilisons l'implémentation de Poly1305 optimisée pour micro-contrôleur du projet `cifra`[4], et l'implémentation de GMAC de la librairie `mbed TLS`[5]. L'implémentation de Poly1305 n'utilise pas de table, alors que l'implémentation de GCM utilise des tables par morceaux de 1 bit. Nous utilisons aussi une implémentation en assembleur de Chaskey (sur 12 tours) de B. Haase, optimisée pour Cortex-M0[6] comme

4. https://github.com/ctz/cifra
5. https://tls.mbed.org/
6. http://mouha.be/wp-content/uploads/chaskey_cortex_m0.zip

Table 5.1 – Évaluation de la multiplication dans divers corps.
Nous rapportons les temps de calcul en cycles/multiplication et en cycles/byte (cycles/octet) (pour comparer différentes tailles de corps), sur deux micro-contrôleurs, un Cortex-M0+ et un Cortex-M4.

Corps	**Implémentation**	**Cortex-M0+**		**Cortex-M4**	
		C/mul	**C/B**	**C/mul**	**C/B**
$\mathbb{F}_{2^{130}-5}$	Poly1305 (cifra)	4609	288	3022	189
$\mathbb{F}_{2^{128}}$	GHASH(mbedtls)	2060	129	2746	172
N/A[†]	Chaskey (B. Haase)	218	14	229	14
$\mathbb{F}_{2^{128}}$	1-bit chunks	2360	148	2050	128
	4-bit chunks	768	48	562	35
	8-bit chunks	-	-	305	19
$\mathbb{F}_{2^{64}}$	1-bit chunks	726	91	676	85
	4-bit chunks	170	21	153	19
	8-bit chunks	97	12	88	11

[†]Chaskey n'utilise pas de multiplications ; nous rapportons le temps pour évaluer la fonction de tour sur 128 bits.

point de comparaison. Chaskey est l'une des primitives cryptographiques les plus rapides sur micro-contrôleur, tel que le mesure le projet FELICS [Din+15] [7].

Nous avons implémenté des multiplications par tables en C et en assembleur, en utilisant plusieurs tailles de morceaux (1 bit, 4 bits, et 8 bits, ces version sont appelées respectivement 1-bit chunks, 4-bit chunks et 8-bit chunks dans la suite). Notons que nous n'avons pas pu implémenter la multiplication dans $\mathbb{F}_{2^{128}}$ par morceaux de 8 bits sur le Cortex-M0+ car les tables sont trop grandes pour la mémoire de ce petit micro-contrôleur.

Discussion. Nos résultats d'implémentation montrent que l'implémentation par table peut être très rapide sur micro-contrôleur, en consacrant une partie de la mémoire aux tables. En particulier, la multiplication sur $\mathbb{F}_{2^{64}}$ avec des morceaux de 4 ou 8 bits est compétitive avec Chaskey. L'utilisation de table limite la possibilité de changer de clef, mais les micro-contrôleurs ne sont pas prévus pour communiquer simultanément avec un grand nombre de clients utilisant des clefs différentes en même temps. De plus, certains micro-contrôleurs n'ont pas de multiplicateur matériel, ou en ont un peu efficace (certains Cortex-M0+ prennent 32 cycles pour une multiplication de 32 bits).

Nous voyons aussi que la multiplication dans un corps plus petit est plus rapide (en cycles/octet), comme prévu. Ceci confirme qu'un MAC basé sur une fonction de hachage sur 64 bits couplée à un mode au delà de la borne des anniversaires est plus rapide qu'un MAC basé sur une fonction de hachage sur 128 bits utilisant le mode Wegman-Carter-Shoup.

Nous avons décidé d'utiliser un corps binaire car cela permet de réutiliser du code ou du matériel prévu pour GCM (comme la multiplication sans retenue disponible sur la plupart des CPU Intel et ARM – mais pas sur les micro-contrôleurs actuels).

Construction des tables de multiplication. Le calcul de la table de multiplication doit être effectuée pour chaque clef (*i.e.* un calcul par branche). La façon de faire directe est de hacher le message branche par branche, et avant de hacher une branche, de calculer sa table de multiplication.

À l'inverse, on peut construire les tables de multiplication en une seule fois et on peut considérer cela comme un précalcul. Ceci peut être envisagé si beaucoup de mémoire est disponible.

7. https://www.cryptolux.org/index.php/FELICS_Block_Ciphers_Brief_Results

Table 5.2 – Temps de calcul et mémoire pour le calcul de la table de multiplication d'une clef.

Taille d'état	Version	Cycles		Mémoire
		Cortex-M0+	Cortex-M4	
128 bits	1-bit chunks	3984	2756	4KB
	4-bit chunks	16992	10918	8KB
	8-bit chunks	-	104922	64KB
64 bits	1-bit chunks	1440	1131	1KB
	4-bit chunks	6144	3769	2KB
	8-bit chunks	53184	40142	16KB

En revanche, notre construction XPoly utilise plusieurs clefs, et nous ne pouvons pas stocker un grand nombre de tables sur un micro-contrôleur. C'est pourquoi nous supposons qu'une implémentation ne précalculerait que la table de k_1, et construirait les autres tables à la volée lorsque nécessaire. Ceci permet de hacher des messages courts efficacement en n'utilisant que k_1, et pour des messages plus longs le temps de construction des tables est amorti car chaque table est utilisée pour λ multiplications.

Afin d'évaluer une valeur acceptable de λ, nous avons effectué des mesures de la construction des tables, tels que donnés en table 5.2. Calculer une table de multiplication est relativement coûteux ; en particulier avec des morceaux de 8 bits il nous faut remplir une table de 2048 valeurs sur 64 bits (respectivement 4096 valeurs sur 128 bits) pour implémenter la multiplication sur $\mathbb{F}_{2^{64}}$ (respectivement $\mathbb{F}_{2^{128}}$).

En se fondant sur ces résultats d'évaluation, nous avons décidé d'utiliser $\lambda = 1024$, afin que le temps passé à calculer les tables de multiplication soit petit, mais tel que la perte de sécurité soit raisonnable pour un long message.

5.5.3 Une instanciation concrète : XPMAC

Nous pouvons désormais définir une construction de MAC concrète en s'appuyant sur nos analyses et comparer ses performances à celles d'autres constructions de MAC. Tel qu'expliqué précédemment, nous utilisons la famille de fonctions de hachage universelle XPoly sur le corps $\mathbb{F}_{2^{64}}$ avec $\lambda = 1024$.

Pour gérer les messages de longueur variable, il nous faut un schéma de rembourrage (padding). Pour faire simple, nous rembourrons le dernier bloc de 64 bits avec des zéros si nécessaire, et nous ajoutons un bloc de 64 bits contenant la longueur du message.

Pour la finalisation de la construction de MAC, nous utilisons la construction $F(H(M)\|N)$ de WMAC. Après comparaison de divers choix pour la PRF, nous avons décidé d'utiliser Noekeon [Dae+00], un chiffrement par blocs sur 128 bits avec une implémentation très efficace sur micro-contrôleur. Nous concaténons la haché de 64 bits avec un nonce de 64 bits, chiffrons le tout, puis tronquons la sortie à 64 bits. Nous utilisons aussi Noekeon pour dériver les sous-clefs utilisées dans XPoly à partir de la clef du chiffrement par bloc, en chiffrant un compteur et en le tronquant à 64 bits : $k_i = [\text{Noekeon}(i\|0)]_{1\ldots64}$. Plus précisément, on chiffre i concaténé à 64 zéros aux bits de poids faible, ce qui permet d'avoir une séparation de domaines entre Noekeon appliqué au nonce (le nonce sur 64 bits est placé dans les bits de poids faible de l'entrée de Noekeon) et Noekeon appliqué au compteur (le compteur est placé dans les 64 bits de poids fort). Pour valider cette séparation de domaine, nous imposons simplement que le nonce N soit non-nul.

Calculons maintenant la sécurité de XPMAC. Notons $n = 64$ la taille du MAC, et l_{max} la taille d'entrée maximale.

Considérons tout d'abord une version de XPMAC avec des clefs indépendantes, notée XPMAC$^{\$}$. On a que XPoly est $\frac{\lambda}{2^n}$-AU. Notons Noekeon$' : x \in \{0,1\}^{2n} \mapsto [\text{Noekeon}(x)]_{1\ldots n}$,

c'est-à-dire la version que nous utilisons, partant de $2n$ bits et tronquée à n bits.

$$\begin{aligned}\mathbf{Adv}^{\mathrm{MAC}}_{\mathsf{XPMAC}^{\$}}(q) &= \mathbf{Adv}^{\mathrm{MAC}}_{\mathrm{WMAC}[\mathsf{XPoly},\mathrm{Noekeon}']}(q)\\ &\leqslant \mathbf{Adv}^{\mathrm{PRF}}_{\mathrm{Noekeon}'}(q) + \tfrac{\lambda}{2^n} + \tfrac{1}{2^n}\\ &\leqslant \mathbf{Adv}^{\mathrm{PRF}}_{\mathrm{Noekeon}}(q) + \tfrac{\lambda}{2^n} + \tfrac{1}{2^n}\\ &\leqslant \mathbf{Adv}^{\mathrm{PRP}}_{\mathrm{Noekeon}}(q) + \tfrac{q^2}{2^{2n}} + \tfrac{\lambda+1}{2^n}\,,\end{aligned}$$

où la dernière ligne est obtenue par le lemme de changement PRP-PRF. On peut raffiner cette borne en utilisant le théorème 4.2 de [BI99], qui nous dit que :

$$\mathbf{Adv}^{\mathrm{PRF}}_{\mathrm{Noekeon}'}(q) \leqslant \mathbf{Adv}^{\mathrm{PRP}}_{\mathrm{Noekeon}}(q) + \tfrac{\sqrt{q}}{2^n}\,,$$

On obtient donc :

$$\mathbf{Adv}^{\mathrm{MAC}}_{\mathsf{XPMAC}^{\$}}(q) \leqslant \mathbf{Adv}^{\mathrm{PRP}}_{\mathrm{Noekeon}}(q) + \tfrac{\sqrt{q}}{2^n} + \tfrac{\lambda+1}{2^n}\,.$$

Considérons maintenant XPMAC dans sa vraie version, *i.e.* avec des clefs $k_i = [\mathrm{Noekeon}(i)]_{1\ldots n}$, $1 \leqslant i \leqslant \frac{l_{max}}{\lambda}$. On a :

$$\begin{aligned}\mathbf{Adv}^{\mathrm{MAC}}_{\mathsf{XPMAC}}(q) &\leqslant \mathbf{Adv}^{\mathrm{MAC}}_{\mathsf{XPMAC}^{\$}}(q) + \mathbf{Adv}^{\mathrm{PRF}}_{\mathrm{Noekeon}'}\left(\tfrac{l_{max}}{\lambda}\right)\\ &\leqslant \mathbf{Adv}^{\mathrm{PRP}}_{\mathrm{Noekeon}}(q) + \tfrac{\sqrt{q}}{2^n} + \tfrac{\lambda+1}{2^n} + \mathbf{Adv}^{\mathrm{PRP}}_{\mathrm{Noekeon}}\left(\tfrac{l_{max}}{\lambda}\right) + \tfrac{\sqrt{l_{max}}}{\sqrt{\lambda}2^n}\\ &= \mathbf{Adv}^{\mathrm{PRP}}_{\mathrm{Noekeon}}\left(q + \tfrac{l_{max}}{\lambda}\right) + \tfrac{\sqrt{q}+\sqrt{\frac{l_{max}}{\lambda}}}{2^n} + \tfrac{\lambda+1}{2^n}\,.\end{aligned}$$

De façon subsidiaire, on peut déduire la borne en cas de réutilisation de nonces, auquel cas la borne de sécurité de WMAC est réduite à celle de Hacher-puis-PRF :

$$\mathbf{Adv}^{\mathrm{MAC}}_{\mathsf{XPMAC}\ \not{nonce}}(q) \leqslant \mathbf{Adv}^{\mathrm{PRP}}_{\mathrm{Noekeon}}\left(q + \tfrac{l_{max}}{\lambda}\right) + \tfrac{\sqrt{q}+\sqrt{\frac{l_{max}}{\lambda}}}{2^n} + \tfrac{\frac{q^2}{2}\lambda+1}{2^n}\,.$$

En particulier, si l'adversaire respecte les nonces, XPMAC est sûr tant que $q < 2^n$. Remarquons que même pour $q = 2^n$, l'avantage de l'adversaire reste nettement inférieur à 1.

Pour comparaison, voici l'avantage d'un attaquant pour GCM [Ber05a] :

$$\mathbf{Adv}^{\mathrm{MAC}}_{\mathrm{GCM}}(q) <= \mathbf{Adv}^{\mathrm{PRP}}_{\mathrm{AES}}(q) + \tfrac{l_{max}}{2^{2n}}\left(1 - \tfrac{q}{2^{2n}}\right)^{-\frac{q+1}{2}}\,.$$

Résultats d'évaluation de XPMAC. La figure 5.8 montre les résultats d'évaluation de XPMAC avec diverses tailles de message sur un Cortex-M4. Nous voyons que le temps croît presque linéairement avec la taille de message, avec du temps supplémentaire pour calculer les tables de multiplication tous les 8KB.

Nous comparons aussi nos résultats d'implémentation avec d'autres primitives. Lorsqu'il est implémenté avec des morceaux de 8 bits, XPMAC réussit à être compétitif avec Chaskey, qui est pourtant un MAC pour micro-contrôleur très agressif et ce qui était l'objectif de ce travail. Mieux encore, par rapport à Chaskey, notre construction basée sur les fonctions de hachage universelles offre de meilleurs arguments de sécurité, en ne se reposant que sur la sécurité de Noekeon, un chiffrement par blocs bien établi. La comparaison avec l'implémentation de GMAC de `mbed TLS` est aussi très intéressante. Cette implémentation utilise des tables par morceaux de 1 bit, mais reste deux fois plus lente que XPMAC avec des morceaux de 1 à cause de la taille de corps plus grande.

Variantes possibles. Il est tout à fait envisageable de remplacer Noekeon par AES ou un autre chiffrement par blocs de 128 bits (pour une implémentation sur micro-contrôleur 32 bits, Noekeon semble adapté a priori).

Nous nous sommes intéressés en particulier au cas du MAC ci-dessus utilisant un chiffrement par blocs sur 128 bits, mais il existe d'autres options intéressantes suivant les primitives disponibles.

En particulier, si un chiffrement par blocs de 64 bits à bas coût est déjà implémenté, nous recommandons d'utiliser la construction EWCDM pour implémenter un MAC basé sur XPoly. Ceci peut être combiné avec un chiffrement en utilisant CENC [Iwa06] afin d'obtenir un mode de chiffrement authentifié à bas coût avec une sécurité jusqu'à 2^{64} blocs de messages.

Alternativement, un MAC déterministe utilisant la construction HF avec un haché sur 128 bits peut aussi être intéressant afin d'éviter de devoir gérer un nonce (le coût de calcul sera plus élevé du fait de la plus grande taille de corps, mais restera similaire à celui de GMAC).

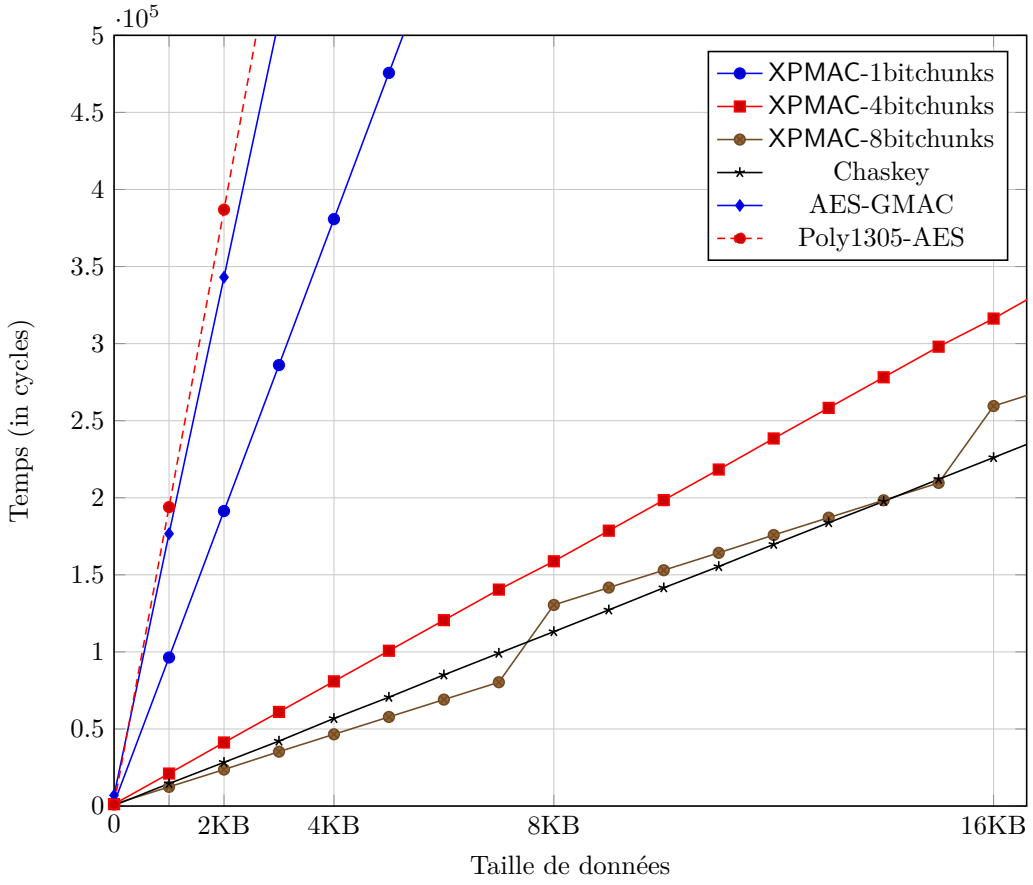


Figure 5.8 – Graphe Temps/Taille des données pour XPMAC et d'autres MACs sur un Cortex-M4.

5.6 Conclusion sur XPMAC

Dans ce travail, nous avons revisité les algorithmes de MAC basés sur les fonctions de hachage universelles dans le contexte de la cryptographie à bas coût. En particulier,

nous montrons qu'il existe des constructions avec des propriétés bien meilleures que celles de la primitive GMAC qui est largement utilisée, et nous en donnons une instanciation concrète dont nous mesurons les performances. Cette analyse de performance montre qu'il est possible de créer un MAC basé sur les fonctions de hachage universelles, donc avec une sécurité au sens de la théorie de l'information, avec une efficacité similaire à celle de Chaskey sur des micro-contrôleurs de type Cortex-M.

Chapitre 6

Perspectives

Ce fut une belle balade

L'objectif de cette thèse était d'étudier des stratégies pour baisser les coûts de la cryptographie tout en garantissant la sécurité. Ces objectifs ont été atteints dans divers cas – les boîtes-S, les matrices de diffusion et les MACs – grâce à des approches innovantes.

Ces travaux ont requis des techniques variées, allant de l'analyse mathématique de fonctions booléennes à l'implémentation sur micro-contrôleurs, en passant par des algorithmes variés, de la combinatoire, des probabilités, des statistiques et de l'algèbre.

En fin de compte, le résultat est là, puisque ces travaux m'ont permis d'obtenir les meilleurs compromis entre coût d'implémentation et sécurité pour les boîtes-S et les matrices de diffusion de la littérature, et le MAC à bas coût le plus performant qui ait des preuves de sécurité robustes, XPMAC.

Bien évidemment, comprendre comment résister aux attaques nécessite de se placer parfois en position d'attaquant, ce que j'ai fait pour l'analyse de FLIP, et ce qui a permis par la suite que soit développée une version sûre de FLIP, qui reste l'un des chiffrements les plus adaptés pour le chiffrement homomorphe hybride.

Grâce à ces travaux, j'ai désormais exploré un large panorama de la cryptographie symétrique et j'ai été en mesure de comprendre les choix qui ont été faits par le passé et, avec le recul, d'identifier de nombreuses pistes de recherches. Si j'ai exploité certaines de ces pistes, je projette d'en étudier beaucoup d'autres : conception de chiffrements complets, études d'attaques sur les implémentations matérielles ou développement de nouvelles techniques d'analyse de la sécurité des chiffrements, par exemple.

Je développe dans ce chapitre les grandes lignes de quelques-unes de ces idées.

6.1 Boîtes-S

6.1.0.1 Prise de recul

Les récents progrès sur la construction de boîtes-S, que ce soient ceux de Li et Wang [LW14], la création des Papillons [PUB16] ou les études de ma thèse, offre une approche neuve de la structure des boîtes-S.

En effet, si l'idée de départ d'utiliser les réseaux de Feistel, de type MISTY ou de type Lai-Massey était de construire des boîtes-S à bas coût, on voit désormais que cette approche permet de surcroît d'obtenir des boîtes-S avec des propriétés cryptographiques aussi bonnes que les constructions envisagées précédemment.

Il était naturel après ces études d'imaginer des combinaisons de réseaux de Feistel et de type MISTY, or c'est à ce moment que Perrin *et al.* ont développé les Papillons, une

structure qui effectivement combine des opérations de type Feistel et de type MISTY et qui atteint des propriétés excellentes. Fait intéressant, l'étude des Feistel et des MISTY avait pour objectif le bas coût alors que l'étude des Papillons avait pour but la résistance cryptographique, pourtant ces travaux ont convergé, ce qui tendrait à contredire l'idée générale qu'il faut un compromis entre le bas coût et la résistance des boîtes-S.

6.1.0.2 Perspectives

Les deux axes de recherche sur les boîtes-S étudiés dans cette thèse sont la résistance cryptographique et le bas coût, et si les études coïncident à l'heure actuelle, les approches pour les études futures peuvent être différentes.

Résistance cryptographique. On se rappellera qu'il existe deux grands problèmes sur la résistance des boîtes-S : le grand problème APN : on ne sait pas construire des permutations optimalement résistantes aux attaques différentielles en dimension paire, à part la boîte-S de Dillon (ou le Papillon), et on ne sait pas s'il est possible d'atteindre une linéarité inférieure à $2^{\frac{n}{2}+1}$ en dimension paire.

Les Papillons atteignent les meilleures bornes actuelles, et parviennent même à être l'exception visée par le grand problème APN. De plus, ces Papillons sont extrêmement structurés, ce qui rend leur étude abordable, et l'existence d'équivalences entre le Papillon ouvert et la Papillon fermé simplifie encore cette étude. Les Papillons semblent donc une alternative prometteuse par rapport aux fonctions puissances, jusqu'ici étudiées pour obtenir de bonnes propriétés cryptographiques. Si les généralisations du Papillon n'ont pas permis d'atteindre de meilleure sécurité pour le moment, de nouvelles approches de généralisation en seront peut-être capables.

Bas coût. À l'heure actuelle, la structure permettant les meilleurs compromis entre le coût d'implémentation des boîtes-S et la sécurité est le réseau de Feistel.

Les travaux menés dans la section 2.2 offrent un grand nombre de pistes possibles pour poursuivre ces études.

À titre d'exemple, il peut être intéressant d'étudier les réseaux de Feistel, de type MISTY et les variantes de type Lai-Massey avec une structure déséquilibrée, puisqu'on remarque que, dans le cas des réseaux de type MISTY, ceci permet un réel progrès dans la sécurité obtenue sans trop augmenter le coût d'implémentation (ou d'analyse).

Une autre piste pourrait être d'étudier des constructions sur $2n$ bits, avec $n \neq 4$, puisque la dimension 4 est particulière : elle ne permet pas d'avoir des permutations avec d'excellentes propriétés cryptographiques (il n'y a pas de permutation APN sur 4 bits). Concrètement, la meilleure boîte-S connue sur 8 bits (l'inverse) a des propriétés cryptographiques équivalentes aux meilleures permutations sur 4 bits. Ainsi, il peut être envisageable, comme on l'a fait pour passer de 4 à 8 bits, de passer de 8 à 16 bits en obtenant une uniformité différentielle de 8 et une linéarité de 64. Ceci permettrait d'atteindre directement une meilleure sécurité sur 16 bits par la boîte-S (plutôt que sur 8 bits) et ainsi on pourrait réduire le nombre de tours (à peu près de moitié) pour la même sécurité.

Une autre voie qui me semble prometteuse est de construire des fonctions sur mn bits à partir de fonctions sur n bits. Pour les chiffrements en particulier, cela a déjà été envisagé par les réseaux de Feistel généralisés.

Cette approche est difficile par deux aspects : il existe de nombreuses façons de généraliser, et il est difficile d'analyser a posteriori la sécurité de fonctions sur mn bits dès que mn est plus grand que 20. Il s'agit donc d'une approche exploratoire dans laquelle il

serait néanmoins nécessaire de pouvoir étudier la sécurité par construction plutôt que de construire des boîtes-S et de tester leur sécurité ensuite.

L'avantage majeur de travaux dans ce sens est qu'on serait alors capable de construire et d'étudier très précisément des permutations sur mn bits, avec mn potentiellement grand : une telle permutation utilisée comme boîte-S permettrait de faire glisser une grande part de l'analyse vers des boîtes-S bien comprises plutôt que sur des modèles tels que la wide-trail strategy qui demande des suppositions et des approximations pour gérer la présence de la clef. De plus, les permutations en dimension mn très grande sont utilisées dans des primitives cryptographiques variées, pas uniquement dans les chiffrements par blocs, et un tel travail aurait donc un impact large.

Idéalement, il serait souhaitable d'exhiber une structure de construction de permutations sur mn bits qui permette d'étudier les propriétés cryptographiques *théoriquement* (en exhibant des formules plutôt qu'en devant calculer l'uniformité différentielle et la linéarité après coup), et qui dans le même temps permette d'atteindre de bonnes propriétés cryptographiques.

Ainsi, en découpant une permutation de mn bits en opérations sur n bits, on serait capable d'obtenir des arguments de sécurité précis pour une grande permutation sur mn bits alors que l'implémentation serait peu coûteuse : il n'y aurait à implémenter que des fonctions sur n bits.

Dans ce sens, une généralisation des Papillons sur mn bits avec des opérations sur n bits me semble prometteuse. En effet, dans le cas du Papillon, on est capable d'étudier très précisément non seulement l'uniformité différentielle et la linéarité, mais aussi les distributions complètes, et ce pour n'importe quelle dimension de la forme $2 \times (2t + 1)$. Ces études n'ont été effectuées que pour le cas $m = 2$, mais il n'est peut-être pas impossible d'envisager d'analyser très précisément une généralisation du Papillon à une structure sur mn bits, $m > 2$.

6.2 Fonctions de diffusion

Si nos travaux sur les matrices de diffusion permettent d'exhiber des matrices optimalement résistantes (MDS) en garantissant qu'elles sont les moins coûteuses de leur classe (une classe pourtant large), on pourrait penser qu'on approche de matrices de diffusion optimales.

On n'aurait pas entièrement tort, pourtant il reste des pistes à explorer.

Par exemple, il est dorénavant envisageable d'essayer d'obtenir une même matrice de diffusion M avec une bonne résistance (si possible MDS), mais qui ait de nombreuses implémentations possibles afin qu'on puisse choisir l'implémentation optimale en fonction du cas d'application. Ceci serait un grand pas pour construire des chiffrements par blocs sûrs, et en même temps peu coûteux pour tous les cas d'application, en particulier cela permettrait d'envisager un standard de chiffrement pour un faisceau d'applications encore plus large qu'AES.

À l'opposé, on pourrait vouloir obtenir des implémentations plus adaptées à certains cas d'usage. Par exemple, nos constructions en circuit ne prennent pas en compte des instructions disponibles sur un grand nombre de CPUs, comme des XORs sur un grand nombre de bits à la fois. Utiliser de telles instructions pourrait permettre d'obtenir des matrices MDS encore moins chères suivant le jeu d'instructions disponibles.

6.3 Analyse de sécurité des chiffrements par blocs

Bien que que je n'aie publié aucun travail sur le sujet, je pense qu'il est aujourd'hui nécessaire d'avoir une réflexion de fond sur le modèle d'analyse que nous utilisons pour les chiffrements par blocs.

Spécifiquement, le modèle le plus couramment utilisé est la wide-trail strategy, ou d'autres modèles d'analyse par comptage de boîtes-S actives, qui permettent de réduire assez simplement les critères de résistance d'un chiffrement par blocs aux attaques différentielles et linéaires à des critères sur les boîtes-S et les matrices de diffusion.

Or, à mon sens, ce modèle d'analyse est désormais mature et commence à montrer ses limites. En effet, il existe aujourd'hui une quantité loin d'être négligeable de chiffrements par blocs aux propriétés variées, qui utilisent le comptage de boîtes-S actives pour argumenter leur résistance aux attaques classiques. Malheureusement, on voit aussi que ce modèle ne suffit pas. Par exemple, en comptant les boîtes-S actives, les auteurs du chiffrement KLEIN [GNL11] ont montré que, d'après le modèle, ce chiffrement devait être capable de résister aux attaques différentielles. Pourtant il existe une attaque différentielle sur KLEIN [LN15].

Ces modèles d'analyse ont plusieurs inconvénients majeurs. Le premier est qu'ils se reposent sur des hypothèses fausses, comme par exemple que les tours sont indépendants les uns des autres (ce qui serait vrai si les clefs de tours étaient des variables aléatoires indépendantes, mais ce n'est jamais le cas). Le deuxième est que ces modèles d'analyse sont très approximatifs.

En effet, observons simplement le cas des arguments de sécurité contre les attaques différentielles. Le comptage de boîtes-S actives permet uniquement de borner la probabilité des caractéristiques différentielles, sous hypothèse que les tours sont indépendants. Or, nous sommes intéressés par la distribution des différentielles d'une permutation. Tentons d'estimer l'écart entre ce que nous avons et ce que nous voulons.

Lorsqu'on fait l'hypothèse que les tours sont indépendants et qu'on ne s'intéresse qu'aux caractéristiques différentielles, la première conséquence est que rajouter des tours est toujours positif pour la résistance de la permutation. À l'évidence, cela n'a pas de sens : pour une permutation aléatoire P, composer P avec autre permutation aléatoire P' une permutation $P \circ P'$ qui a une chance sur deux d'être plus résistante que P aux attaques différentielles.

Bien évidemment, si la permutation de départ P est faible face aux attaques différentielles, puisqu'en moyenne les permutations sont relativement résistantes aux attaques différentielles, appliquer une opération P' sur P a de grandes chances d'améliorer la résistance de la permutation.

Mais dès que P n'est plus trop faible, composer P avec une opération quelle qu'elle soit n'a pas de raison particulière d'améliorer la sécurité de P.

Ainsi si le modèle d'analyse permet de répéter une fonction de tour un grand nombre de fois pour atteindre un bon niveau de sécurité face aux attaques différentielles, il faut se remémorer que dès qu'on s'intéresse aux différentielles plutôt qu'aux caractéristiques ou qu'on retire l'hypothèse que les tours sont indépendants, cette analyse n'a plus beaucoup de sens.

De plus, même si l'on était effectivement capable d'estimer la probabilité des caractéristiques d'une grande permutation, cela ne serait même pas un argument valable de sécurité face aux attaques différentielles.

En effet, une attaque différentielle exploite une différentielle et non une caractéristique. Or, si l'on étudie en pratique comment une différentielle se décompose en caractéristiques, on observe que parfois, les différentielles les plus probables sont formées d'une agglomération de caractéristiques différentielles très peu probables. Dans ce type de situation, garantir

que les caractéristiques différentielles sont peu probables n'apporte aucune information sur la probabilité des différentielles, qui est l'information qui nous intéresse.

Enfin, je tiens à noter que toutes ces études tentent d'étudier le maximum des probabilités des [caractéristiques] différentielles. Or, souvenons-nous que l'objectif pour un chiffrement n'est pas d'avoir des [caractéristiques] différentielles de probabilité faible, mais d'avoir une distribution de la table des différences proche de celle d'une permutation aléatoire. Puisqu'une permutation aléatoire ne minimise pas la probabilité des [caractéristiques] différentielles, étudier le maximum n'a pas de sens en particulier ; il s'agit simplement d'une quantité plus simple à mesurer que la distribution dans son ensemble.

Étudier une autre statistique telle que l'espérance ou la variance plutôt que le maximum ne serait pas moins cohérent et pourrait permettre de donner des informations supplémentaires.

Ainsi, je pense que nous sommes désormais à une étape importante de l'étude des chiffrements symétriques. Le modèle que nous avons utilisé depuis plusieurs décennies est aujourd'hui mature et nous sommes en mesure de voir ses limites. Je pense qu'il est temps, avec l'expérience acquise, d'imaginer un nouveau modèle d'analyse plus précis qui permette de donner de meilleurs arguments de sécurité, au moins pour les attaques classiques

Une piste possible serait d'étudier des « boîtes-S » sur de grandes tailles ($\gg$ 8 bits), *i.e.* de grandes fonctions non-linéaires sans clef. En effet, l'absence de clef permet que l'étude de telles fonctions, même sur de grandes tailles, soit plus simple que l'étude de chiffrements. Dans le même temps, connaître des résultats sur des critères de résistance aux attaques précis, tels que l'uniformité différentielle et la linéarité, pour des boîtes-S très grandes, serait avantageux. Cela permettrait de mieux maîtriser ce qui se passe dans le chiffrement par rapport à certaines façons de faire actuelles qui consistent à itérer un grand nombre de tours faibles et compter sur le fait que l'analyse est difficile. En effet, l'objectif en cryptographie est de rendre les attaques impossibles, mais de rendre la cryptanalyse simple, et mieux maîtriser de grandes boîtes-S permettrait de simplifier l'analyse.

Dans ce cadre, il paraît néanmoins essentiel de conserver des constructions de grandes boîte-S structurées pour permettre une implémentation peu coûteuse. De plus, il faut que ces grandes boîtes-S soient construites à partir d'opérations peu coûteuses, donc d'opérations sur peu de bits. Il s'agit donc d'un projet ambitieux, mais des idées similaires à des réseaux de Feistel généralisés paraissent une piste raisonnable.

Bien évidemment, il ne s'agit ici que de simples idées, mais il me paraît aujourd'hui nécessaire de tourner une page dans l'étude des chiffrements par blocs, car le modèle d'analyse actuel ne permet plus à la recherche sur les chiffrements par blocs de progresser vers de meilleurs arguments de sécurité.

Bibliographie

[AA04] Dmitri Asonov et Rakesh Agrawal. “Keyboard Acoustic Emanations”. In : *2004 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, mai 2004, p. 3–11.

[AB12] Jean-Philippe Aumasson et Daniel J. Bernstein. “SipHash : A Fast Short-Input PRF”. In : *INDOCRYPT 2012*. Sous la dir. de Steven D. Galbraith et Mridul Nandi. T. 7668. LNCS. Springer, Heidelberg, déc. 2012, p. 489–508.

[AF13] Daniel Augot et Matthieu Finiasz. “Exhaustive search for small dimension recursive MDS diffusion layers for block ciphers and hash functions”. In : *Proceedings of the 2013 IEEE International Symposium on Information Theory, Istanbul, Turkey, July 7-12, 2013*. 2013, p. 1551–1555.

[AK92] Ibrahim A Al-Kadit. “Origins of cryptology : The Arab contributions”. In : *Cryptologia* 16.2 (1992), p. 97–126.

[Alb+14] Martin R. Albrecht, Benedikt Driessen, Elif Bilge Kavun, Gregor Leander, Christof Paar et Tolga Yalçın. “Block Ciphers - Focus on the Linear Layer (feat. PRIDE)”. In : *CRYPTO 2014, Part I*. Sous la dir. de Juan A. Garay et Rosario Gennaro. T. 8616. LNCS. Springer, Heidelberg, août 2014, p. 57–76.

[Alb+15] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen et Michael Zohner. “Ciphers for MPC and FHE”. In : *EUROCRYPT 2015, Part I*. Sous la dir. d’Elisabeth Oswald et Marc Fischlin. T. 9056. LNCS. Springer, Heidelberg, avr. 2015, p. 430–454.

[AO97] KaZumaro Aoki et Kazuo Ohta. “Strict Evaluation of the Maximum Average of Differential Probability and the Maximum Average of Linear Probability”. In : *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences* E80A.1 (1997), p. 2–8.

[Ban+15] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita et Francesco Regazzoni. “Midori : A Block Cipher for Low Energy”. In : *ASIACRYPT 2015, Part II*. Sous la dir. de Tetsu Iwata et Jung Hee Cheon. T. 9453. LNCS. Springer, Heidelberg, 2015, p. 411–436.

[Bar+10] Paulo Barreto, Ventzislav Nikov, Svetla Nikova, Vincent Rijmen et Elmar Tischhauser. “Whirlwind : a new cryptographic hash function”. In : *Designs, Codes and Cryptography* 56.2 (2010), p. 141–162. issn : 1573-7586.

[BBR16] Subhadeep Banik, Andrey Bogdanov et Francesco Regazzoni. “Atomic-AES : A Compact Implementation of the AES Encryption/Decryption Core”. In : *INDOCRYPT 2016*. Sous la dir. d’Orr Dunkelman et Somitra Kumar Sanadhya. T. 10095. LNCS. Springer, Heidelberg, déc. 2016, p. 173–190.

[BC09] John Black et Martin Cochran. “MAC Reforgeability”. In : *FSE 2009*. Sous la dir. d’Orr Dunkelman. T. 5665. LNCS. Springer, Heidelberg, fév. 2009, p. 345–362.

[BCK96] Mihir Bellare, Ran Canetti et Hugo Krawczyk. “Keying Hash Functions for Message Authentication”. In : *CRYPTO’96*. Sous la dir. de Neal Koblitz. T. 1109. LNCS. Springer, Heidelberg, août 1996, p. 1–15.

[BD08] Steve Babbage et Matthew Dodd. “The MICKEY Stream Ciphers”. In : *New Stream Cipher Designs - The eSTREAM Finalists*. Sous la dir. de Matthew J. B. Robshaw et Olivier Billet. T. 4986. Lecture Notes in Computer Science. Springer, 2008, p. 191–209. isbn : 978-3-540-68350-6.

[BD94] Thomas Beth et Cunsheng Ding. “On Almost Perfect Nonlinear Permutations”. In : *EUROCRYPT’93*. Sous la dir. de Tor Helleseth. T. 765. LNCS. Springer, Heidelberg, mai 1994, p. 65–76.

[Bea+13] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks et Louis Wingers. *The SIMON and SPECK Families of Lightweight Block Ciphers*. Cryptology ePrint Archive, Report 2013/404. http://eprint.iacr.org/2013/404. 2013.

[Bei+16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich et Siang Meng Sim. “The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS”. In : *CRYPTO 2016, Part II*. Sous la dir. de Matthew Robshaw et Jonathan Katz. T. 9815. LNCS. Springer, Heidelberg, août 2016, p. 123–153.

[Ben+09] Ryad Benadjila, Olivier Billet, Henri Gilbert, Gilles Macario-Rat, Thomas Peyrin, Matt Robshaw et Yannick Seurin. “Sha-3 proposal : ECHO”. In : *Submission to NIST (updated)* (2009), p. 113.

[Ben+13] Ryad Benadjila, Jian Guo, Victor Lomné et Thomas Peyrin. *Implementing Lightweight Block Ciphers on x86 Architectures*. Cryptology ePrint Archive, Report 2013/445. http://eprint.iacr.org/2013/445. 2013.

[Ber+06] Thierry P. Berger, Anne Canteaut, Pascale Charpin et Yann Laigle-Chapuy. “On Almost Perfect Nonlinear Functions Over $\mathbf{F}_2^n$”. In : *IEEE Trans. Information Theory* 52.9 (2006), p. 4160–4170.

[Ber+08] Côme Berbain, Olivier Billet, Anne Canteaut, Nicolas Courtois, Henri Gilbert, Louis Goubin, Aline Gouget, Louis Granboulan, Cédric Lauradoux, Marine Minier, Thomas Pornin et Hervé Sibert. “Sosemanuk, a Fast Software-Oriented Stream Cipher”. In : *New Stream Cipher Designs - The eSTREAM Finalists*. Sous la dir. de Matthew J. B. Robshaw et Olivier Billet. T. 4986. Lecture Notes in Computer Science. Springer, 2008, p. 98–118. isbn : 978-3-540-68350-6.

[Ber05a] Daniel J. Bernstein. “Stronger Security Bounds for Wegman-Carter-Shoup Authenticators”. In : *EUROCRYPT 2005*. Sous la dir. de Ronald Cramer. T. 3494. LNCS. Springer, Heidelberg, mai 2005, p. 164–180.

[Ber05b] Daniel J. Bernstein. “The Poly1305-AES Message-Authentication Code”. In : *FSE 2005*. Sous la dir. d’Henri Gilbert et Helena Handschuh. T. 3557. LNCS. Springer, Heidelberg, fév. 2005, p. 32–49.

[Ber08] Daniel J. Bernstein. “The Salsa20 Family of Stream Ciphers”. In : *New Stream Cipher Designs - The eSTREAM Finalists*. Sous la dir. de Matthew J. B. Robshaw et Olivier Billet. T. 4986. Lecture Notes in Computer Science. Springer, 2008, p. 84–97. isbn : 978-3-540-68350-6.

[BI99] M. Bellare et R. Impagliazzo. *A tool for obtaining tighter security analyses of pseudorandom function based constructions, with applications to PRP to PRF conversion*. Cryptology ePrint Archive, Report 1999/024. https://eprint.iacr.org/1999/024. 1999.

[Bir+03] Alex Biryukov, Christophe De Cannière, An Braeken et Bart Preneel. "A Toolbox for Cryptanalysis : Linear and Affine Equivalence Algorithms". In : *EUROCRYPT 2003*. Sous la dir. d'Eli Biham. T. 2656. LNCS. Springer, Heidelberg, mai 2003, p. 33–50.

[BKL16] Christof Beierle, Thorsten Kranz et Gregor Leander. "Lightweight Multiplication in GF(2^n) with Applications to MDS Matrices". In : *CRYPTO 2016, Part I*. Sous la dir. de Matthew Robshaw et Jonathan Katz. T. 9814. LNCS. Springer, Heidelberg, août 2016, p. 625–653.

[BL08] Marcus Brinkmann et Gregor Leander. "On the classification of APN functions up to dimension five". In : *Des. Codes Cryptography* 49.1-3 (2008), p. 273–288.

[Bla+99] John Black, Shai Halevi, Hugo Krawczyk, Ted Krovetz et Phillip Rogaway. "UMAC : Fast and Secure Message Authentication". In : *CRYPTO'99*. Sous la dir. de Michael J. Wiener. T. 1666. LNCS. Springer, Heidelberg, août 1999, p. 216–233.

[Bla00] John R Black. "Message authentication codes". Thèse de doct. University of California, Davis, 2000.

[BMP13] Joan Boyar, Philip Matthews et René Peralta. "Logic Minimization Techniques with Applications to Cryptology". In : *Journal of Cryptology* 26.2 (avr. 2013), p. 280–312.

[BN14] Céline Blondeau et Kaisa Nyberg. "Links between Truncated Differential and Multidimensional Linear Properties of Block Ciphers and Underlying Attack Complexities". In : *EUROCRYPT 2014*. Sous la dir. de Phong Q. Nguyen et Elisabeth Oswald. T. 8441. LNCS. Springer, Heidelberg, mai 2014, p. 165–182.

[Bog+07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin et C. Vikkelsoe. "PRESENT : An Ultra-Lightweight Block Cipher". In : *CHES 2007*. Sous la dir. de Pascal Paillier et Ingrid Verbauwhede. T. 4727. LNCS. Springer, Heidelberg, sept. 2007, p. 450–466.

[Bor+12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knežević, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen et Tolga Yalçın. "PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications - Extended Abstract". In : *ASIACRYPT 2012*. Sous la dir. de Xiaoyun Wang et Kazue Sako. T. 7658. LNCS. Springer, Heidelberg, déc. 2012, p. 208–225.

[Bos+16] Erik Boss, Vincent Grosso, Tim Güneysu, Gregor Leander, Amir Moradi et Tobias Schneider. "Strong 8-bit Sboxes with Efficient Masking in Hardware". In : *CHES 2016*. Sous la dir. de Benedikt Gierlichs et Axel Y. Poschmann. T. 9813. LNCS. Springer, Heidelberg, août 2016, p. 171–193.

[BP10] Joan Boyar et René Peralta. "A New Combinational Logic Minimization Technique with Applications to Cryptology". In : *SEA*. T. 6049. Lecture Notes in Computer Science. Springer, 2010, p. 178–189.

[BR+00] PSLM Barreto, Vincent Rijmen et al. “The Whirlpool hashing function”. In : *First open NESSIE Workshop, Leuven, Belgium*. T. 13. Citeseer. 2000, p. 14.

[BR00] John Black et Phillip Rogaway. “CBC MACs for Arbitrary-Length Messages : The Three-Key Constructions”. In : *CRYPTO 2000*. Sous la dir. de Mihir Bellare. T. 1880. LNCS. Springer, Heidelberg, août 2000, p. 197–215.

[BR02] John Black et Phillip Rogaway. “A Block-Cipher Mode of Operation for Parallelizable Message Authentication”. In : *EUROCRYPT 2002*. Sous la dir. de Lars R. Knudsen. T. 2332. LNCS. Springer, Heidelberg, 2002, p. 384–397.

[Bra+07] Carl Bracken, Eimear Byrne, Nadya Markin et Gary McGuire. “Determining the Nonlinearity of a New Family of APN Functions”. In : *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes - AAECC-17*. T. 4851. LNCS. Springer, 2007, p. 72–79.

[Bro+10] K.A. Browning, J.F. Dillon, M.T. McQuistan et A.J. Wolfe. “An APN permutation in dimension six”. In : *Finite Fields : Theory and Applications - FQ9*. T. 518. Contemporary Mathematics. AMS, 2010, p. 33–42.

[BRS67] E.R. Berlekamp, H. Rumsey et G. Solomon. “On the solution of algebraic equations over finite fields”. In : *Inform. Contr.* 12.5 (1967), p. 553–564.

[BS91] Eli Biham et Adi Shamir. “Differential Cryptanalysis of DES-like Cryptosystems”. In : *CRYPTO’90*. Sous la dir. d’Alfred J. Menezes et Scott A. Vanstone. T. 537. LNCS. Springer, Heidelberg, août 1991, p. 2–21.

[BS97] Eli Biham et Adi Shamir. “Differential Fault Analysis of Secret Key Cryptosystems”. In : *CRYPTO’97*. Sous la dir. de Burton S. Kaliski Jr. T. 1294. LNCS. Springer, Heidelberg, août 1997, p. 513–525.

[BVZ08] Martin Boesgaard, Mette Vesterager et Erik Zenner. “The Rabbit Stream Cipher”. In : *New Stream Cipher Designs - The eSTREAM Finalists*. Sous la dir. de Matthew J. B. Robshaw et Olivier Billet. T. 4986. Lecture Notes in Computer Science. Springer, 2008, p. 69–83. isbn : 978-3-540-68350-6.

[BW72] Elwyn R. Berlekamp et Lloyd R. Welch. “Weight distributions of the cosets of the (32, 6) Reed-Muller code”. In : *IEEE Transactions on Information Theory* 18.1 (1972), p. 203–207.

[Can+00] Anne Canteaut, Claude Carlet, Pascale Charpin et Caroline Fontaine. “Propagation Characteristics and Correlation-Immunity of Highly Nonlinear Boolean Functions”. In : *EUROCRYPT 2000*. Sous la dir. de Bart Preneel. T. 1807. LNCS. Springer, Heidelberg, mai 2000, p. 507–522.

[Can+01a] Anne Canteaut, Claude Carlet, Pascale Charpin et Caroline Fontaine. “On cryptographic properties of the cosets of R(1, m)”. In : *IEEE Trans. Information Theory* 47.4 (2001), p. 1494–1513.

[Can+01b] Anne Canteaut, Claude Carlet, Pascale Charpin et Caroline Fontaine. “On cryptographic properties of the cosets of R(1, m)”. In : *IEEE Trans. Information Theory* 47.4 (2001), p. 1494–1513.

[Can+16] Anne Canteaut, Sergiu Carpov, Caroline Fontaine, Tancrède Lepoint, María Naya-Plasencia, Pascal Paillier et Renaud Sirdey. “Stream Ciphers : A Practical Solution for Efficient Homomorphic-Ciphertext Compression”. In : *FSE 2016*. Sous la dir. de Thomas Peyrin. T. 9783. LNCS. Springer, Heidelberg, mar. 2016, p. 313–333.

[Can06] Anne Canteaut. “Analysis and design of secret-key ciphers”. Habilitation à diriger des recherches. Université Pierre et Marie Curie - Paris VI, sept. 2006.

[Car10] Claude Carlet. “Boolean Models and Methods in Mathematics, Computer Science, and Engineering”. In : Cambridge University Press, 2010. Chap. Boolean functions for cryptography and error correcting codes, p. 257–397.

[CCD00] Anne Canteaut, Pascale Charpin et Hans Dobbertin. “Binary m-sequences with three-valued crosscorrelation : A proof of Welch’s conjecture”. In : *IEEE Trans. Information Theory* 46.1 (2000), p. 4–9.

[CCZ98] Claude Carlet, Pascale Charpin et Victor A. Zinoviev. “Codes, Bent Functions and Permutations Suitable For DES-like Cryptosystems”. In : *Des. Codes Cryptography* 15.2 (1998), p. 125–156.

[CD96] Thomas W. Cusick et Hans Dobbertin. “Some new three-valued crosscorrelation functions for binary m-sequences”. In : *IEEE Trans. Information Theory* 42.4 (1996), p. 1238–1240.

[CDL16] Anne Canteaut, Sébastien Duval et Gaëtan Leurent. “Construction of Lightweight S-Boxes Using Feistel and MISTY Structures”. In : *SAC 2015*. Sous la dir. d’Orr Dunkelman et Liam Keliher. T. 9566. LNCS. Springer, Heidelberg, août 2016, p. 373–393.

[CDP17] Anne Canteaut, Sébastien Duval et Léo Perrin. “A Generalisation of Dillon’s APN Permutation With the Best Known Differential and Nonlinear Properties for All Fields of Size 2^{4k+2}”. In : *IEEE Trans. Information Theory* 63.11 (2017), p. 7575–7591.

[Cha+99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao et Pankaj Rohatgi. “Towards Sound Approaches to Counteract Power-Analysis Attacks”. In : *CRYPTO’99*. Sous la dir. de Michael J. Wiener. T. 1666. LNCS. Springer, Heidelberg, août 1999, p. 398–412.

[Che+13] Jung Hee Cheon, Jean-Sébastien Coron, Jinsu Kim, Moon Sung Lee, Tancrède Lepoint, Mehdi Tibouchi et Aaram Yun. “Batch Fully Homomorphic Encryption over the Integers”. In : *EUROCRYPT 2013*. Sous la dir. de Thomas Johansson et Phong Q. Nguyen. T. 7881. LNCS. Springer, Heidelberg, mai 2013, p. 315–335.

[CLS17] Benoît Cogliati, Jooyoung Lee et Yannick Seurin. “New Constructions of MACs from (Tweakable) Block Ciphers”. In : *IACR Trans. Symm. Cryptol.* 2017.2 (2017), p. 27–58. issn : 2519-173X.

[CMR17] Claude Carlet, Pierrick Méaux et Yann Rotella. “Boolean functions with restricted input and their robustness ; application to the FLIP cipher”. In : *IACR Trans. Symm. Cryptol.* 2017.3 (2017), p. 192–227. issn : 2519-173X.

[CP08] Christophe De Cannière et Bart Preneel. “Trivium”. In : *New Stream Cipher Designs - The eSTREAM Finalists*. Sous la dir. de Matthew J. B. Robshaw et Olivier Billet. T. 4986. Lecture Notes in Computer Science. Springer, 2008, p. 244–266. isbn : 978-3-540-68350-6.

[CR16] Anne Canteaut et Yann Rotella. “Attacks Against Filter Generators Exploiting Monomial Mappings”. In : *FSE 2016*. Sous la dir. de Thomas Peyrin. T. 9783. LNCS. Springer, Heidelberg, mar. 2016, p. 78–98.

[CS16] Benoît Cogliati et Yannick Seurin. “EWCDM : An Efficient, Beyond-Birthday Secure, Nonce-Misuse Resistant MAC”. In : *CRYPTO 2016, Part I*. Sous la dir. de Matthew Robshaw et Jonathan Katz. T. 9814. LNCS. Springer, Heidelberg, août 2016, p. 121–149.

[CV95] Florent Chabaud et Serge Vaudenay. “Links Between Differential and Linear Cryptanalysis”. In : *EUROCRYPT’94*. Sous la dir. d’Alfredo De Santis. T. 950. LNCS. Springer, Heidelberg, mai 1995, p. 356–365.

[CW77] J Lawrence Carter et Mark N Wegman. “Universal classes of hash functions”. In : *Proceedings of the ninth annual ACM symposium on Theory of computing*. ACM. 1977, p. 106–112.

[Dae+00] Joan Daemen, Michaël Peeters, Gilles Van Assche et Vincent Rijmen. “Nessie proposal : NOEKEON”. In : *First Open NESSIE Workshop*. 2000.

[Dat+17] Nilanjan Datta, Avijit Dutta, Mridul Nandi, Goutam Paul et Liting Zhang. “Single Key Variant of PMAC_Plus”. In : *IACR Trans. Symm. Cryptol.* 2017.4 (2017), p. 268–305. issn : 2519-173X.

[Des] *Data Encryption Standard*. National Bureau of Standards, NBS FIPS PUB 46, U.S. Department of Commerce. Jan. 1977.

[DH76] Whitfield Diffie et Martin E. Hellman. “New Directions in Cryptography”. In : *IEEE Transactions on Information Theory* 22.6 (1976), p. 644–654.

[DH79] Whitfield Diffie et Martin E Hellman. “Privacy and authentication : An introduction to cryptography”. In : *Proceedings of the IEEE* 67.3 (1979), p. 397–427.

[DHS14] Yarkin Doröz, Yin Hu et Berk Sunar. “Homomorphic AES Evaluation using NTRU”. In : *IACR Cryptology ePrint Archive* 2014 (2014), p. 39.

[DHT17] Wei Dai, Viet Tung Hoang et Stefano Tessaro. *Information-theoretic Indistinguishability via the Chi-squared Method*. Cryptology ePrint Archive, Report 2017/537. http://eprint.iacr.org/2017/537. 2017.

[Die+92] M. Dietzfelbinger, J. Gil, Y. Matias et N. Pippenger. “Polynomial hash functions are reliable”. In : *Automata, Languages and Programming*. Sous la dir. de W. Kuich. Berlin, Heidelberg : Springer Berlin Heidelberg, 1992, p. 235–246. isbn : 978-3-540-47278-0.

[Dij59] Edsger Wybe Dijkstra. “A Note on Two Problems in Connexion with Graphs”. In : *Numerische Mathematik* 1 (1959), p. 269–271.

[Din+15] Daniel Dinu, Alex Biryukov, Johann Grossschädl, Dmitry Khovratovich, YL Corre et Léo Perrin. “Felics–fair evaluation of lightweight cryptographic systems”. In : *NIST Workshop on Lightweight Cryptography*. T. 128. 2015.

[DL18a] Sébastien Duval et Gaëtan Leurent. “Lightweight MACs from Universal Hash Functions”. In : *En soumission* (2018).

[DL18b] Sébastien Duval et Gaëtan Leurent. “MDS Matrices with Lightweight Circuits”. In : *IACR Transactions on Symmetric Cryptology* 2018.2 (2018), p. 48–78. issn : 2519-173X.

[DLR16] Sébastien Duval, Virginie Lallemand et Yann Rotella. “Cryptanalysis of the FLIP Family of Stream Ciphers”. In : *CRYPTO 2016, Part I*. Sous la dir. de Matthew Robshaw et Jonathan Katz. T. 9814. LNCS. Springer, Heidelberg, août 2016, p. 457–475.

[Dob01] Hans Dobbertin. “Almost Perfect Nonlinear Power Functions on GF(2 n) : A New Case for n Divisible by 5”. In : *Finite Fields and Applications*. Sous la dir. de Dieter Jungnickel et Harald Niederreiter. Springer Berlin Heidelberg, 2001, p. 113–121. isbn : 978-3-642-56755-1.

[Dob98] Hans Dobbertin. “One-to-One Highly Nonlinear Power Functions on GF(2^n)”. In : *Appl. Algebra Eng. Commun. Comput.* 9.2 (1998), p. 139–152.

[Dob99a] Hans Dobbertin. “Almost Perfect Nonlinear Power Functions on GF(2n) : The Niho Case”. In : *Inf. Comput.* 151.1-2 (1999), p. 57–72.

[Dob99b] Hans Dobbertin. “Almost Perfect Nonlinear Power Functions on GF(2^n) : The Welch Case”. In : *IEEE Trans. Information Theory* 45.4 (1999), p. 1271–1275.

[Dor+14] Yarkin Doröz, Aria Shahverdi, Thomas Eisenbarth et Berk Sunar. “Toward Practical Homomorphic Evaluation of Block Ciphers Using Prince”. In : *Financial Cryptography and Data Security - FC 2014 Workshops, BITCOIN and WAHC 2014, Christ Church, Barbados, March 7, 2014, Revised Selected Papers*. 2014, p. 208–220.

[DR01] Joan Daemen et Vincent Rijmen. “The Wide Trail Design Strategy”. In : *8th IMA International Conference on Cryptography and Coding*. Sous la dir. de Bahram Honary. T. 2260. LNCS. Springer, Heidelberg, déc. 2001, p. 222–238.

[DR02] Joan Daemen et Vincent Rijmen. *The Design of Rijndael : AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002. isbn : 3-540-42580-2.

[Ehr+78] William F Ehrsam, Carl HW Meyer, John L Smith et Walter L Tuchman. *Message verification and transmission error detection by block chaining*. US Patent 4,074,066. 1978.

[EJ00] Patrik Ekdahl et Thomas Johansson. “SNOW – a new stream cipher”. In : *PROCEEDINGS OF FIRST OPEN NESSIE WORKSHOP, KU-LEUVEN*. 2000.

[FFW17] Shihui Fu, Xiutao Feng et Baofeng Wu. “Differentially 4-Uniform Permutations with the Best Known Nonlinearity from Butterflies”. In : *IACR Trans. Symm. Cryptol.* 2017.2 (2017), p. 228–249. issn : 2519-173X.

[FIPS113] *Computer Data Authentication*. National Bureau of Standards, NIST FIPS PUB 113, U.S. Department of Commerce. 1985.

[GC91] Henri Gilbert et Guy Chassé. “A Statistical Attack of the FEAL-8 Cryptosystem”. In : *CRYPTO’90*. Sous la dir. d’Alfred J. Menezes et Scott A. Vanstone. T. 537. LNCS. Springer, Heidelberg, août 1991, p. 22–33.

[GMS74] Edgar N Gilbert, F Jessie MacWilliams et Neil JA Sloane. “Codes which detect deception”. In : *Bell Labs Technical Journal* 53.3 (1974), p. 405–424.

[GNL11] Zheng Gong, Svetla Nikova et Yee Wei Law. “KLEIN : A New Family of Lightweight Block Ciphers”. In : *RFID. Security and Privacy - 7th International Workshop, RFIDSec 2011, Amherst, USA, June 26-28, 2011, Revised Selected Papers*. 2011, p. 1–18.

[Gol68] Robert Gold. “Maximal recursive sequences with 3-valued recursive cross-correlation functions (Corresp.)” In : *IEEE Trans. Information Theory* 14.1 (1968), p. 154–156.

[GPP11] Jian Guo, Thomas Peyrin et Axel Poschmann. “The PHOTON Family of Lightweight Hash Functions”. In : *CRYPTO 2011*. Sous la dir. de Phillip Rogaway. T. 6841. LNCS. Springer, Heidelberg, août 2011, p. 222–239.

[Gro+14] Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, Kerem Varici, François Durvaux, Lubos Gaspar et Stéphanie Kerckhof. “SCREAM & iSCREAM Side-Channel Resistant Authenticated Encryption with Masking”. In : *CAESAR competition* (2014).

[Gro+15] Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert et Kerem Varici. “LS-Designs : Bitslice Encryption for Efficient Masked Software Implementations”. In : *FSE 2014*. Sous la dir. de Carlos Cid et Christian Rechberger. T. 8540. LNCS. Springer, Heidelberg, mar. 2015, p. 18–37.

[Guo+11] Jian Guo, Thomas Peyrin, Axel Poschmann et Matthew J. B. Robshaw. "The LED Block Cipher". In : *CHES 2011*. Sous la dir. de Bart Preneel et Tsuyoshi Takagi. T. 6917. LNCS. Springer, Heidelberg, 2011, p. 326–341.

[Göl15] Faruk Göloglu. "Almost perfect nonlinear trinomials and hexanomials". In : *Finite Fields and Their Applications* 33 (2015), p. 258–282.

[Hel+08] Martin Hell, Thomas Johansson, Alexander Maximov et Willi Meier. "The Grain Family of Stream Ciphers". In : *New Stream Cipher Designs - The eSTREAM Finalists*. Sous la dir. de Matthew J. B. Robshaw et Olivier Billet. T. 4986. Lecture Notes in Computer Science. Springer, 2008, p. 179–190. isbn : 978-3-540-68350-6.

[HNR68] Peter E. Hart, Nils J. Nilsson et Bertram Raphael. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In : *IEEE Trans. Systems Science and Cybernetics* 4.2 (1968), p. 100–107.

[HP08] Helena Handschuh et Bart Preneel. "Key-Recovery Attacks on Universal Hash Function Based MAC Algorithms". In : *CRYPTO 2008*. Sous la dir. de David Wagner. T. 5157. LNCS. Springer, Heidelberg, août 2008, p. 144–161.

[HR00] Philip Hawkes et Gregory G. Rose. "Exploiting Multiples of the Connection Polynomial in Word-Oriented Stream Ciphers". In : *ASIACRYPT 2000*. Sous la dir. de Tatsuaki Okamoto. T. 1976. LNCS. Springer, Heidelberg, déc. 2000, p. 303–316.

[HX01] Henk D.L. Hollmann et Qing Xiang. "A Proof of the Welch and Niho Conjectures on Cross-Correlations of Binary m-Sequences". In : *Finite Fields and Their Applications* 7.2 (2001), p. 253–286. issn : 1071-5797.

[IK03] Tetsu Iwata et Kaoru Kurosawa. "OMAC : One-Key CBC MAC". In : *FSE 2003*. Sous la dir. de Thomas Johansson. T. 2887. LNCS. Springer, Heidelberg, fév. 2003, p. 129–153.

[ISW03] Yuval Ishai, Amit Sahai et David Wagner. "Private Circuits : Securing Hardware against Probing Attacks". In : *CRYPTO 2003*. Sous la dir. de Dan Boneh. T. 2729. LNCS. Springer, Heidelberg, août 2003, p. 463–481.

[Iwa+17] Tetsu Iwata, Kazuhiko Minematsu, Thomas Peyrin et Yannick Seurin. "ZMAC : A Fast Tweakable Block Cipher Mode for Highly Secure Message Authentication". In : *CRYPTO 2017, Part III*. Sous la dir. de Jonathan Katz et Hovav Shacham. T. 10403. LNCS. Springer, Heidelberg, août 2017, p. 34–65.

[Iwa06] Tetsu Iwata. "New Blockcipher Modes of Operation with Beyond the Birthday Bound Security". In : *FSE 2006*. Sous la dir. de Matthew J. B. Robshaw. T. 4047. LNCS. Springer, Heidelberg, mar. 2006, p. 310–327.

[Jea+17] Jérémy Jean, Thomas Peyrin, Siang Meng Sim et Jade Tourteaux. "Optimizing Implementations of Lightweight Building Blocks". In : *IACR Trans. Symm. Cryptol.* 2017.4 (2017), p. 130–168. issn : 2519-173X.

[Kah96] David Kahn. *The Codebreakers : The comprehensive history of secret communication from ancient times to the internet*. Simon et Schuster, 1996.

[Kar16] Pierre Karpman. "Exercice de style". working paper or preprint. Jan. 2016.

[Kas71] Tadao Kasami. "The Weight Enumerators for Several Clauses of Subcodes of the 2nd Order Binary Reed-Muller Codes". In : *Information and Control* 18.4 (1971), p. 369–394.

[Ker83] A Kerckhoffs. "A. Kerckhoffs, la cryptographie militaire, Journal des Sciences Militaires IX, 38 (1883)." In : *Journal des sciences militaires* 9 (1883), p. 38.

[Kho+14] Khoongming Khoo, Thomas Peyrin, Axel York Poschmann et Huihui Yap. "FOAM : Searching for Hardware-Optimal SPN Structures and Components with a Fair Comparison". In : *CHES 2014*. Sous la dir. de Lejla Batina et Matthew Robshaw. T. 8731. LNCS. Springer, Heidelberg, sept. 2014, p. 433–450.

[KJJ99] Paul C. Kocher, Joshua Jaffe et Benjamin Jun. "Differential Power Analysis". In : *CRYPTO'99*. Sous la dir. de Michael J. Wiener. T. 1666. LNCS. Springer, Heidelberg, août 1999, p. 388–397.

[Knu+10] Lars R. Knudsen, Gregor Leander, Axel Poschmann et Matthew J. B. Robshaw. "PRINTcipher : A Block Cipher for IC-Printing". In : *CHES 2010*. Sous la dir. de Stefan Mangard et François-Xavier Standaert. T. 6225. LNCS. Springer, Heidelberg, août 2010, p. 16–32.

[Knu69] Donald E. Knuth. *The Art of Computer Programming, Volume II : Seminumerical Algorithms*. Addison-Wesley, 1969. isbn : 0201038021.

[Koc96] Paul C. Kocher. "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems". In : *CRYPTO'96*. Sous la dir. de Neal Koblitz. T. 1109. LNCS. Springer, Heidelberg, août 1996, p. 104–113.

[Kra+17] Thorsten Kranz, Gregor Leander, Ko Stoffelen et Friedrich Wiemer. "Shorter Linear Straight-Line Programs for MDS Matrices". In : *IACR Trans. Symm. Cryptol.* 2017.4 (2017), p. 188–211. issn : 2519-173X.

[KS07] Liam Keliher et Jiayuan Sui. "Exact maximum expected differential and linear probability for two-round Advanced Encryption Standard". In : *IET Information Security* 1.2 (2007), p. 53–57.

[KS12] Gohar M Kyureghyan et Valentin Suder. "On inverses of APN exponents". In : *IEEE International Symposium on Information Theory - ISIT 2012*. IEEE. 2012, p. 1207–1211.

[Lan90] Philippe Langevin. "Covering radius of RM (1, 9) in RM (3, 9)". In : *International Symposium on Coding Theory and Applications - EUROCODE '90*. Sous la dir. de Gérard D. Cohen et Pascale Charpin. T. 514. LNCS. Springer, 1990, p. 51–59.

[Lea+11] Gregor Leander, Mohamed Ahmed Abdelraheem, Hoda AlKhzaimi et Erik Zenner. "A Cryptanalysis of PRINTcipher : The Invariant Subspace Attack". In : *CRYPTO 2011*. Sous la dir. de Phillip Rogaway. T. 6841. LNCS. Springer, Heidelberg, août 2011, p. 206–221.

[Li+18] Yongqiang Li, Shizhu Tian, Yuyin Yu et Mingsheng Wang. "On the Generalization of Butterfly Structure". In : *IACR Trans. Symm. Cryptol.* 2018.1 (2018), p. 160–179. issn : 2519-173X.

[Lim99] Chae Hoon Lim. "A Revised Version of Crypton - Crypton V1.0". In : *FSE'99*. Sous la dir. de Lars R. Knudsen. T. 1636. LNCS. Springer, Heidelberg, mar. 1999, p. 31–45.

[LLN14] Kristin E. Lauter, Adriana López-Alt et Michael Naehrig. "Private Computation on Encrypted Genomic Data". In : *Progress in Cryptology - LATINCRYPT 2014 - Third International Conference on Cryptology and Information Security in Latin America, Florianópolis, Brazil, September 17-19, 2014, Revised Selected Papers*. 2014, p. 3–27.

[LM91] Xuejia Lai et James L. Massey. "A Proposal for a New Block Encryption Standard". In : *EUROCRYPT'90*. Sous la dir. d'Ivan Damgård. T. 473. LNCS. Springer, Heidelberg, mai 1991, p. 389–404.

[LN14] Tancrède Lepoint et Michael Naehrig. “A Comparison of the Homomorphic Encryption Schemes FV and YASHE”. In : *AFRICACRYPT 14*. Sous la dir. de David Pointcheval et Damien Vergnaud. T. 8469. LNCS. Springer, Heidelberg, mai 2014, p. 318–335.

[LN15] Virginie Lallemand et María Naya-Plasencia. “Cryptanalysis of KLEIN”. In : *FSE 2014*. Sous la dir. de Carlos Cid et Christian Rechberger. T. 8540. LNCS. Springer, Heidelberg, mar. 2015, p. 451–470.

[LN83] Rudolf Lidl et Harald Niederreiter. *Finite fields*. Cambridge university press, 1983.

[LN93] Richard J. Lipton et Jeffrey F. Naughton. “Clocked Adversaries for Hashing”. In : *Algorithmica* 9.3 (1993), p. 239–252.

[LP07a] Gregor Leander et Axel Poschmann. “On the Classification of 4 Bit S-Boxes”. In : *Arithmetic of Finite Fields, First International Workshop, WAIFI 2007, Madrid, Spain, June 21-22, 2007, Proceedings*. 2007, p. 159–176.

[LP07b] Gregor Leander et Axel Poschmann. “On the Classification of 4 Bit S-Boxes”. In : *Arithmetic of Finite Fields - WAIFI*. T. 4547. LNCS. Springer, 2007, p. 159–176.

[LRW02] Moses Liskov, Ronald L. Rivest et David Wagner. “Tweakable Block Ciphers”. In : *CRYPTO 2002*. Sous la dir. de Moti Yung. T. 2442. LNCS. Springer, Heidelberg, août 2002, p. 31–46.

[LS16] Meicheng Liu et Siang Meng Sim. “Lightweight MDS Generalized Circulant Matrices”. In : *FSE 2016*. Sous la dir. de Thomas Peyrin. T. 9783. LNCS. Springer, Heidelberg, mar. 2016, p. 101–120.

[LW14] Yongqiang Li et Mingsheng Wang. “Constructing S-boxes for Lightweight Cryptography with Feistel Structure”. In : *CHES 2014*. Sous la dir. de Lejla Batina et Matthew Robshaw. T. 8731. LNCS. Springer, Heidelberg, sept. 2014, p. 127–146.

[LW16] Yongqiang Li et Mingsheng Wang. “On the Construction of Lightweight Circulant Involutory MDS Matrices”. In : *FSE 2016*. Sous la dir. de Thomas Peyrin. T. 9783. LNCS. Springer, Heidelberg, mar. 2016, p. 121–139.

[LW90] Gilles Lachaud et Jacques Wolfmann. “The weights of the orthogonals of the extended quadratic binary Goppa codes”. In : *IEEE Trans. Information Theory* 36.3 (1990), p. 686–692.

[Mat08] Mitsuru Matsui. “New Structure of Block Ciphers with Provable Security against Differential and Linear Cryptanalysis.” In : *Fast Software Encryption - FSE'96*. T. 1039. LNCS. Springer, 1er jan. 2008, p. 205–218. isbn : 3-540-60865-6.

[Mat94a] Mitsuru Matsui. “Linear Cryptanalysis Method for DES Cipher”. In : *EUROCRYPT'93*. Sous la dir. de Tor Helleseth. T. 765. LNCS. Springer, Heidelberg, mai 1994, p. 386–397.

[Mat94b] Mitsuru Matsui. “The First Experimental Cryptanalysis of the Data Encryption Standard”. In : *CRYPTO'94*. Sous la dir. d'Yvo Desmedt. T. 839. LNCS. Springer, Heidelberg, août 1994, p. 1–11.

[Mat97] Mitsuru Matsui. “New Block Encryption Algorithm MISTY”. In : *FSE'97*. Sous la dir. d'Eli Biham. T. 1267. LNCS. Springer, Heidelberg, jan. 1997, p. 54–68.

[MN17] Bart Mennink et Samuel Neves. “Encrypted Davies-Meyer and Its Dual : Towards Optimal Security Using Mirror Theory”. In : *CRYPTO 2017, Part III*. Sous la dir. de Jonathan Katz et Hovav Shacham. T. 10403. LNCS. Springer, Heidelberg, août 2017, p. 556–583.

[Mou+14] Nicky Mouha, Bart Mennink, Anthony Van Herrewege, Dai Watanabe, Bart Preneel et Ingrid Verbauwhede. “Chaskey : An Efficient MAC Algorithm for 32-bit Microcontrollers”. In : *SAC 2014*. Sous la dir. d’Antoine Joux et Amr M. Youssef. T. 8781. LNCS. Springer, Heidelberg, août 2014, p. 306–323.

[MS77] F. J. MacWilliams et N. J. A. Sloane. *The Theory of Error Correcting Codes*. North Holland, 1977.

[MS87] Richard A. Mollin et Charles Small. “On permutation polynomials over finite fields”. In : *International Journal of Mathematics and Mathematical Sciences* 10.3 (1987), p. 535–543.

[MS89] Willi Meier et Othmar Staffelbach. “Fast Correlation Attacks on Certain Stream Ciphers”. In : *J. Cryptology* 1.3 (1989), p. 159–176.

[MT06] Kazuhiko Minematsu et Yukiyasu Tsunoo. “Provably Secure MACs from Differentially-Uniform Permutations and AES-Based Implementations”. In : *FSE 2006*. Sous la dir. de Matthew J. B. Robshaw. T. 4047. LNCS. Springer, Heidelberg, mar. 2006, p. 226–241.

[MV04] David A. McGrew et John Viega. “The Security and Performance of the Galois/Counter Mode (GCM) of Operation”. In : *INDOCRYPT 2004*. Sous la dir. d’Anne Canteaut et Kapalee Viswanathan. T. 3348. LNCS. Springer, Heidelberg, déc. 2004, p. 343–355.

[Méa+16] Pierrick Méaux, Anthony Journault, François-Xavier Standaert et Claude Carlet. “Towards Stream Ciphers for Efficient FHE with Low-Noise Ciphertexts”. In : *EUROCRYPT 2016, Part I*. Sous la dir. de Marc Fischlin et Jean-Sébastien Coron. T. 9665. LNCS. Springer, Heidelberg, mai 2016, p. 311–343.

[NK95] Kaisa Nyberg et Lars R. Knudsen. “Provable Security Against a Differential Attack”. In : *J. Cryptology* 8.1 (1995), p. 27–37.

[NL15] Y. Nir et A. Langley. *ChaCha20 and Poly1305 for IETF Protocols*. RFC 7539 (Informational). RFC. Fremont, CA, USA : RFC Editor, mai 2015.

[NLV11] Michael Naehrig, Kristin E. Lauter et Vinod Vaikuntanathan. “Can homomorphic encryption be practical ?” In : *Proceedings of the 3rd ACM Cloud Computing Security Workshop, CCSW 2011, Chicago, IL, USA, October 21, 2011*. 2011, p. 113–124.

[Nyb91a] Kaisa Nyberg. “Perfect Nonlinear S-Boxes”. In : *EUROCRYPT’91*. Sous la dir. de Donald W. Davies. T. 547. LNCS. Springer, Heidelberg, avr. 1991, p. 378–386.

[Nyb91b] Kaisa Nyberg. “Perfect nonlinear S-boxes”. In : *Advances in Cryptology - EUROCRYPT’91*. T. 547. LNCS. Springer-Verlag, 1991, p. 378–385.

[Nyb94a] Kaisa Nyberg. “Differentially Uniform Mappings for Cryptography”. In : *EUROCRYPT’93*. Sous la dir. de Tor Helleseth. T. 765. LNCS. Springer, Heidelberg, mai 1994, p. 55–64.

[Nyb94b] Kaisa Nyberg. “Differentially Uniform Mappings for Cryptography”. In : *EUROCRYPT’93*. Sous la dir. de Tor Helleseth. T. 765. LNCS. Springer, Heidelberg, 1994, p. 55–64.

[Nyb96] Kaisa Nyberg. “Generalized Feistel Networks”. In : *ASIACRYPT'96*. Sous la dir. de Kwangjo Kim et Tsutomu Matsumoto. T. 1163. LNCS. Springer, Heidelberg, nov. 1996, p. 91–104.

[Osv02] Dag Arne Osvik. “Speeding up Serpent.” In : *AES Candidate Conference*. 3 jan. 2002, p. 317–329.

[Paa97] C. Paar. “Optimized arithmetic for Reed-Solomon encoders”. In : *Proceedings of IEEE International Symposium on Information Theory*. 1997, p. 250–.

[Pat08] Jacques Patarin. “A Proof of Security in O(2n) for the Xor of Two Random Permutations”. In : *ICITS*. T. 5155. Lecture Notes in Computer Science. Springer, 2008, p. 232–248.

[Pat13] Jacques Patarin. *Security in $O(2^n)$ for the Xor of Two Random Permutations—Proof with the standard H technique*. Cryptology ePrint Archive, Report 2013/368. http://eprint.iacr.org/2013/368. 2013.

[PC14] Gordon Procter et Carlos Cid. “On Weak Keys and Forgery Attacks Against Polynomial-Based MAC Schemes”. In : *FSE 2013*. Sous la dir. de Shiho Moriai. T. 8424. LNCS. Springer, Heidelberg, mar. 2014, p. 287–304.

[PUB16] Léo Perrin, Aleksei Udovenko et Alex Biryukov. “Cryptanalysis of a Theorem : Decomposing the Only Known Solution to the Big APN Problem”. In : *CRYPTO 2016, Part II*. Sous la dir. de Matthew Robshaw et Jonathan Katz. T. 9815. LNCS. Springer, Heidelberg, août 2016, p. 93–122.

[Pv95] Bart Preneel et Paul C. van Oorschot. “MDx-MAC and Building Fast MACs from Hash Functions”. In : *CRYPTO'95*. Sous la dir. de Don Coppersmith. T. 963. LNCS. Springer, Heidelberg, août 1995, p. 1–14.

[Qu+13] Longjiang Qu, Yin Tan, Chik How Tan et Chao Li. “Constructing Differentially 4-Uniform Permutations Over $\mathbf{F}_{2^{2k}}$ via the Switching Method”. In : *IEEE Trans. Information Theory* 59.7 (2013), p. 4675–4686.

[RB08] Matthew J. B. Robshaw et Olivier Billet, éds. *New Stream Cipher Designs - The eSTREAM Finalists*. T. 4986. Lecture Notes in Computer Science. Springer, 2008. isbn : 978-3-540-68350-6.

[Rog99] Phillip Rogaway. “Bucket Hashing and Its Application to Fast Message Authentication”. In : *Journal of Cryptology* 12.2 (1999), p. 91–115.

[Rot76] O. S. Rothaus. “On "Bent" Functions”. In : *J. Comb. Theory, Ser. A* 20.3 (1976), p. 300–305.

[Saa12] Markku-Juhani O. Saarinen. “Cryptographic Analysis of All 4 × 4-Bit S-Boxes”. In : *SAC 2011*. Sous la dir. d'Ali Miri et Serge Vaudenay. T. 7118. LNCS. Springer, Heidelberg, août 2012, p. 118–133.

[Saj+12] Mahdi Sajadieh, Mohammad Dakhilalian, Hamid Mala et Pouyan Sepehrdad. “Recursive Diffusion Layers for Block Ciphers and Hash Functions”. In : *FSE 2012*. Sous la dir. d'Anne Canteaut. T. 7549. LNCS. Springer, Heidelberg, mar. 2012, p. 385–401.

[Sat+01] Akashi Satoh, Sumio Morioka, Kohji Takano et Seiji Munetoh. “A Compact Rijndael Hardware Architecture with S-Box Optimization”. In : *ASIACRYPT 2001*. Sous la dir. de Colin Boyd. T. 2248. LNCS. Springer, Heidelberg, déc. 2001, p. 239–254.

[Sha49] Claude E. Shannon. “Communication theory of secrecy systems”. In : *Bell Systems Technical Journal* 28.4 (1949), p. 656–715.

[Sho96] Victor Shoup. “On Fast and Provably Secure Message Authentication Based on Universal Hashing”. In : *CRYPTO'96*. Sous la dir. de Neal Koblitz. T. 1109. LNCS. Springer, Heidelberg, août 1996, p. 313–328.

[Sho99] Victor Shoup. “Efficient Computation of Minimal Polynomials in Algebraic Extensions of Finite Fields”. In : *Proceedings of the 1999 International Symposium on Symbolic and Algebraic Computation, ISSAC '99, Vancouver, B.C., Canada, July 29-31, 1999*. 1999, p. 53–58.

[Sil00] John R Silvester. “Determinants of block matrices”. In : *The Mathematical Gazette* 84.501 (2000), p. 460–467.

[Sim+15] Siang Meng Sim, Khoongming Khoo, Frédérique E. Oggier et Thomas Peyrin. “Lightweight MDS Involution Matrices”. In : *FSE 2015*. Sous la dir. de Gregor Leander. T. 9054. LNCS. Springer, Heidelberg, mar. 2015, p. 471–493.

[SS16] Sumanta Sarkar et Habeeb Syed. “Lightweight Diffusion Layer : Importance of Toeplitz Matrices”. In : *IACR Trans. Symm. Cryptol.* 2016.1 (2016). http://tosc.iacr.org/index.php/ToSC/article/view/537, p. 95–113. issn : 2519-173X.

[Sti92] Douglas R. Stinson. “Universal Hashing and Authentication Codes”. In : *CRYPTO'91*. Sous la dir. de Joan Feigenbaum. T. 576. LNCS. Springer, Heidelberg, août 1992, p. 74–85.

[TCG92] Anne Tardy-Corfdir et Henri Gilbert. “A Known Plaintext Attack of FEAL-4 and FEAL-6”. In : *CRYPTO'91*. Sous la dir. de Joan Feigenbaum. T. 576. LNCS. Springer, Heidelberg, août 1992, p. 172–181.

[TCT15] Deng Tang, Claude Carlet et Xiaohu Tang. “Differentially 4-uniform bijections by permuting the inverse function”. In : *Des. Codes Cryptography* 77.1 (2015), p. 117–141.

[TGZ16] Yin Tan, Guang Gong et Bo Zhu. “Enhanced criteria on differential uniformity and nonlinearity of cryptographically significant functions”. In : *Cryptography and Communications* 8.2 (2016), p. 291–311.

[Ull+11] Markus Ullrich, Christophe De Cannière, Sebastian Indesteege, Özgül Küçük, Nicky Mouha et Bart Preneel. “Finding Optimal Bitsliced Implementations of 4x4-bit S-Boxes”. In : *SKEW 2011 Symmetric Key Encryption Workshop, Copenhagen, Denmark*. 2011, p. 16–17.

[WC81] Mark N. Wegman et Larry Carter. “New Hash Functions and Their Use in Authentication and Set Equality”. In : *Journal of Computer and System Sciences* 22 (1981), p. 265–279.

[Wu08] Hongjun Wu. “The Stream Cipher HC-128”. In : *New Stream Cipher Designs - The eSTREAM Finalists*. Sous la dir. de Matthew J. B. Robshaw et Olivier Billet. T. 4986. Lecture Notes in Computer Science. Springer, 2008, p. 39–47. isbn : 978-3-540-68350-6.

[WWW13] Shengbao Wu, Mingsheng Wang et Wenling Wu. “Recursive Diffusion Layers for (Lightweight) Block Ciphers and Hash Functions”. In : *SAC 2012*. Sous la dir. de Lars R. Knudsen et Huapeng Wu. T. 7707. LNCS. Springer, Heidelberg, août 2013, p. 355–371.

[WZ11] Wenling Wu et Lei Zhang. “LBlock : A Lightweight Block Cipher”. In : *ACNS 11*. Sous la dir. de Javier Lopez et Gene Tsudik. T. 6715. LNCS. Springer, Heidelberg, juin 2011, p. 327–344.

[Yas10] Kan Yasuda. “The Sum of CBC MACs Is a Secure PRF”. In : *CT-RSA 2010*. Sous la dir. de Josef Pieprzyk. T. 5985. LNCS. Springer, Heidelberg, mar. 2010, p. 366–381.

[Yas11] Kan Yasuda. “A New Variant of PMAC : Beyond the Birthday Bound”. In : *CRYPTO 2011*. Sous la dir. de Phillip Rogaway. T. 6841. LNCS. Springer, Heidelberg, août 2011, p. 596–609.

[Zha+12] Liting Zhang, Wenling Wu, Han Sui et Peng Wang. “3kf9 : Enhancing 3GPP-MAC beyond the Birthday Bound”. In : *ASIACRYPT 2012*. Sous la dir. de Xiaoyun Wang et Kazue Sako. T. 7658. LNCS. Springer, Heidelberg, déc. 2012, p. 296–312.

[Zha+16] Ruoxin Zhao, Baofeng Wu, Rui Zhang et Qian Zhang. *Designing Optimal Implementations of Linear Layers (Full Version)*. Cryptology ePrint Archive, Report 2016/1118. http://eprint.iacr.org/2016/1118. 2016.

[ZHS14] Zhengbang Zha, Lei Hu et Siwei Sun. “Constructing new differentially 4-uniform permutations from the inverse function”. In : *Finite Fields and Their Applications* 25 (2014), p. 64–78.

[De 07] Christophe De Cannière. “Analysis and Design of Symmetric Encryption Algorithms”. Thèse de doct. KU Leuven, 2007.